Faculty of Physics

WARSAW UNIVERSITY OF TECHNOLOGY

ALICE

POLITECHNIKA WARSZAWSKA WARSAW UNIVERSITY OF TECHNOLOGY

Using Machine Learning for Particle Identification in ALICE

Łukasz Graczykowski

Tomasz Trzciński, Kamil Deja, Maja Kabus, Dawid Sitnik, Monika Jakubowska

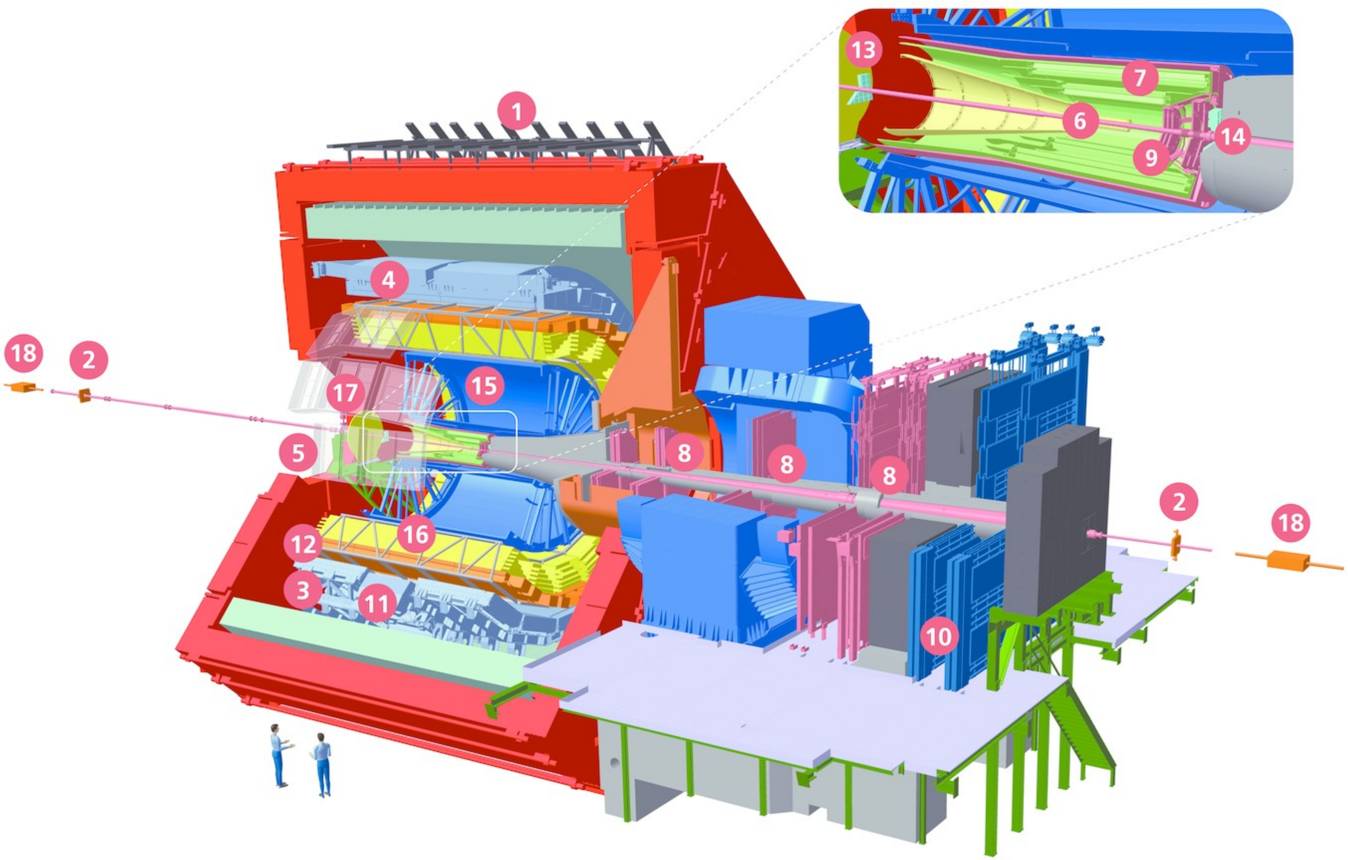
for the ALICE Collaboration

AI4EIC-Exp Workshop

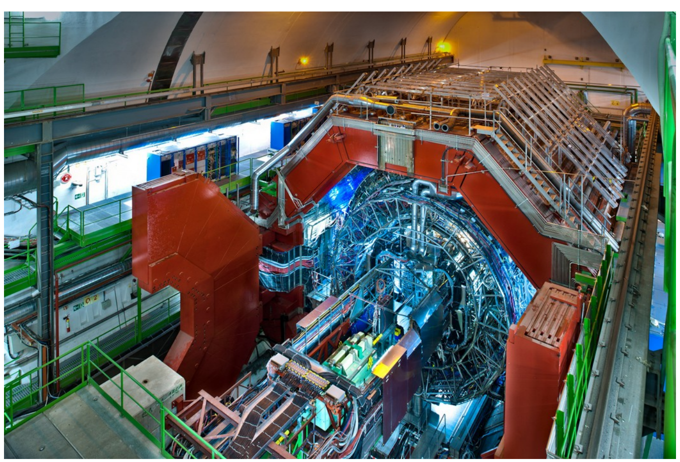
BNL, USA

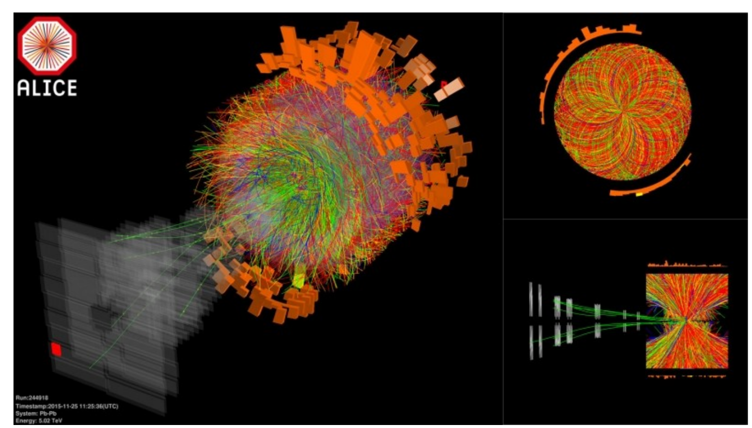
8/09/2021

The ALICE experiment



1. **ACORDE** | ALICE Cosmic Rays Detector
2. **AD** | ALICE Diffractive Detector
3. **DCal** | Di-jet Calorimeter
4. **EMCal** | Electromagnetic Calorimeter
5. **HMPID** | High Momentum Particle Identification Detector
6. **ITS-IB** | Inner Tracking System - Inner Barrel
7. **ITS-OB** | Inner Tracking System - Outer Barrel
8. **MCH** | Muon Tracking Chambers
9. **MFT** | Muon Forward Tracker
10. **MID** | Muon Identifier
11. **PHOS / CPV** | Photon Spectrometer
12. **TOF** | Time Of Flight
13. **T0+A** | Tzero + A
14. **T0+C** | Tzero + C
15. **TPC** | Time Projection Chamber
16. **TRD** | Transition Radiation Detector
17. **V0+** | Vzero + Detector
18. **ZDC** | Zero Degree Calorimeter





Goals of the WUT team

- Use ALICE and its data as a unique environment to advance the Machine Learning field of science
- Identify areas where both ALICE (or HEP in general) and ML communities can mutually benefit
- More focus on Machine Learning research rather than using standard ML tools for ALICE use cases

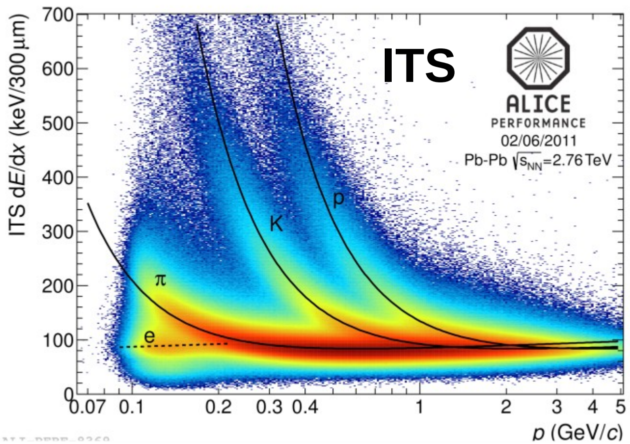
- Disclaimer:
  - I'm a physicist working with ML experts from the WUT IT department
  - My task is to guide and coordinate the work of WUT ML computer scientists within ALICE

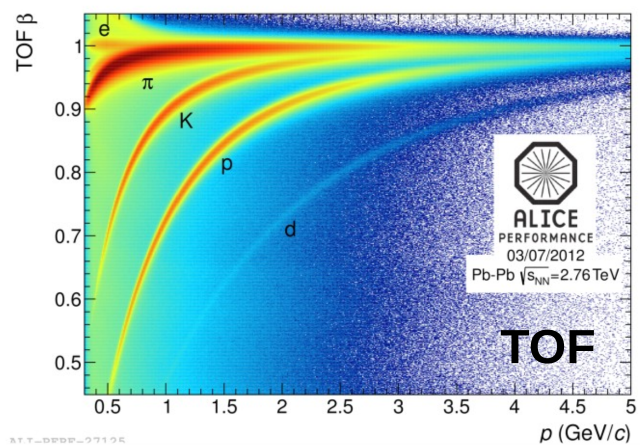
PID with Machine Learning

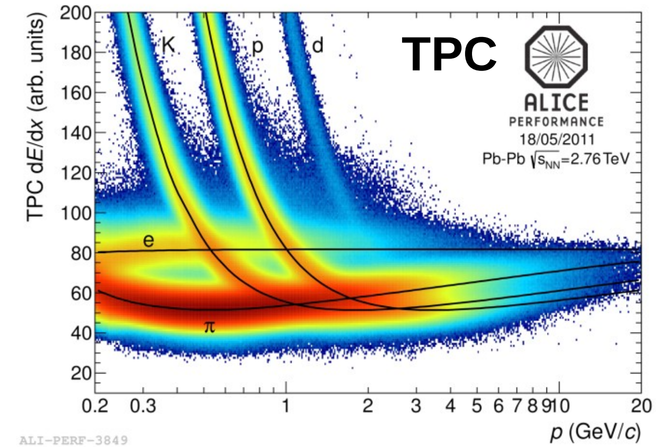
Particle identification

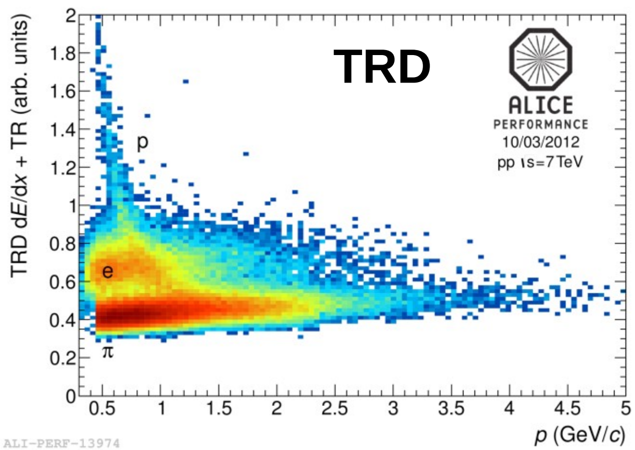
- Particle identification (PID) is one of the most important steps in many physics analyses
- Crucial for Quark-Gluon Plasma measurements
- PID is one of the strongest advantages of ALICE:
  - practically all knows techniques used (dE/dx energy loss, time-of-flight, Cherenkov radiation for hadrons and transition radiation for electrons)
  - possibility to identify (anti-)nuclei
  - very good separation of pions, kaons, protons, electrons over a wide momentum range
  - separation of signals of charged hadrons and electrons for very low momenta (down to 0.1 GeV/c)

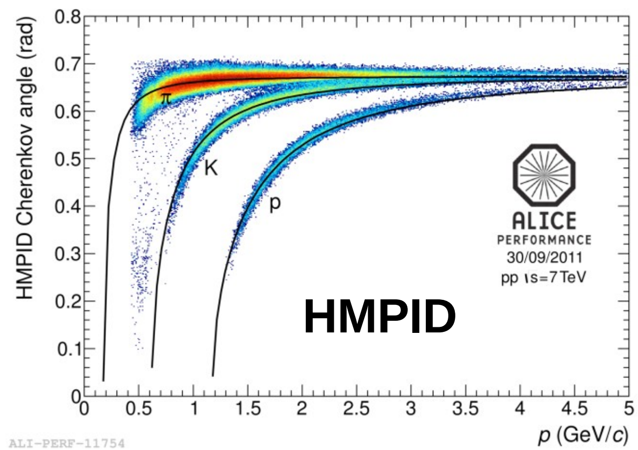
Particle identification

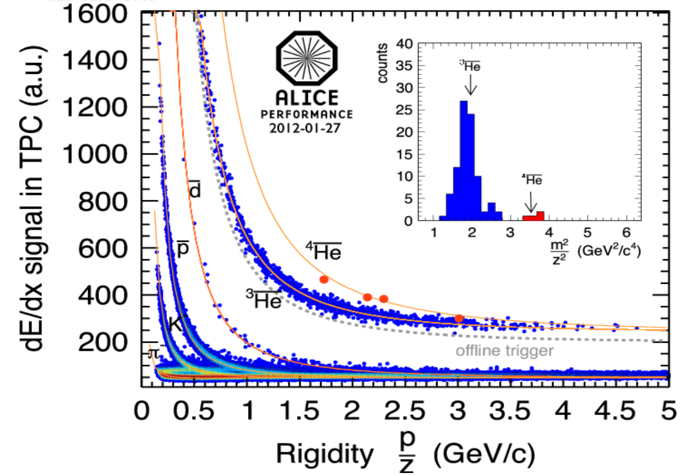


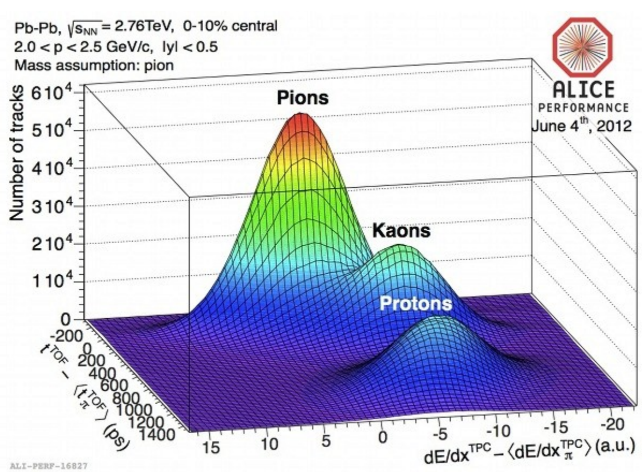






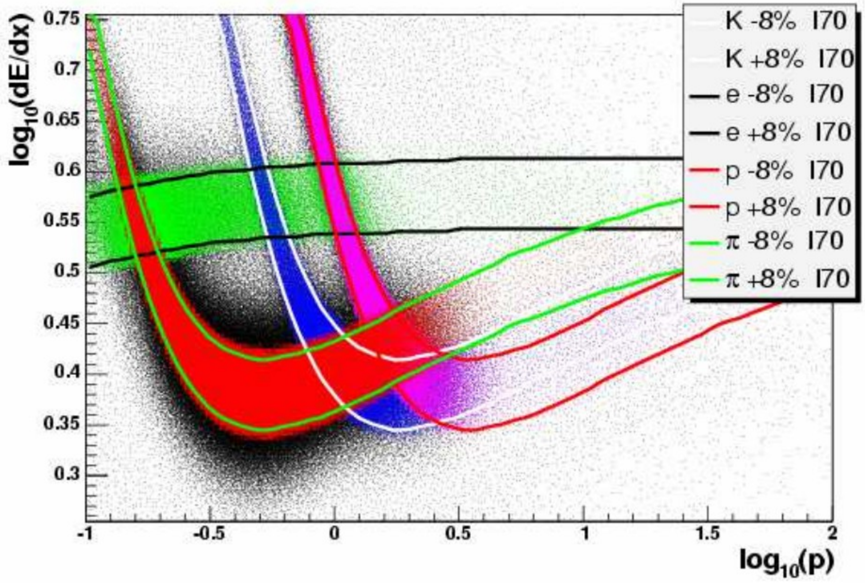






Traditional vs ML PID

- **Traditional PID:**
  - a typical analyzer selects particles "manually" by cutting on certain quantities, like the number of standard deviations of a signal from the expected value (nσ)
  - most limitations come in the regions where signals from different particle species cross
  - "cut" optimization is a time-consuming task



https://arxiv.org/pdf/nucl-ex/0505026.pdf

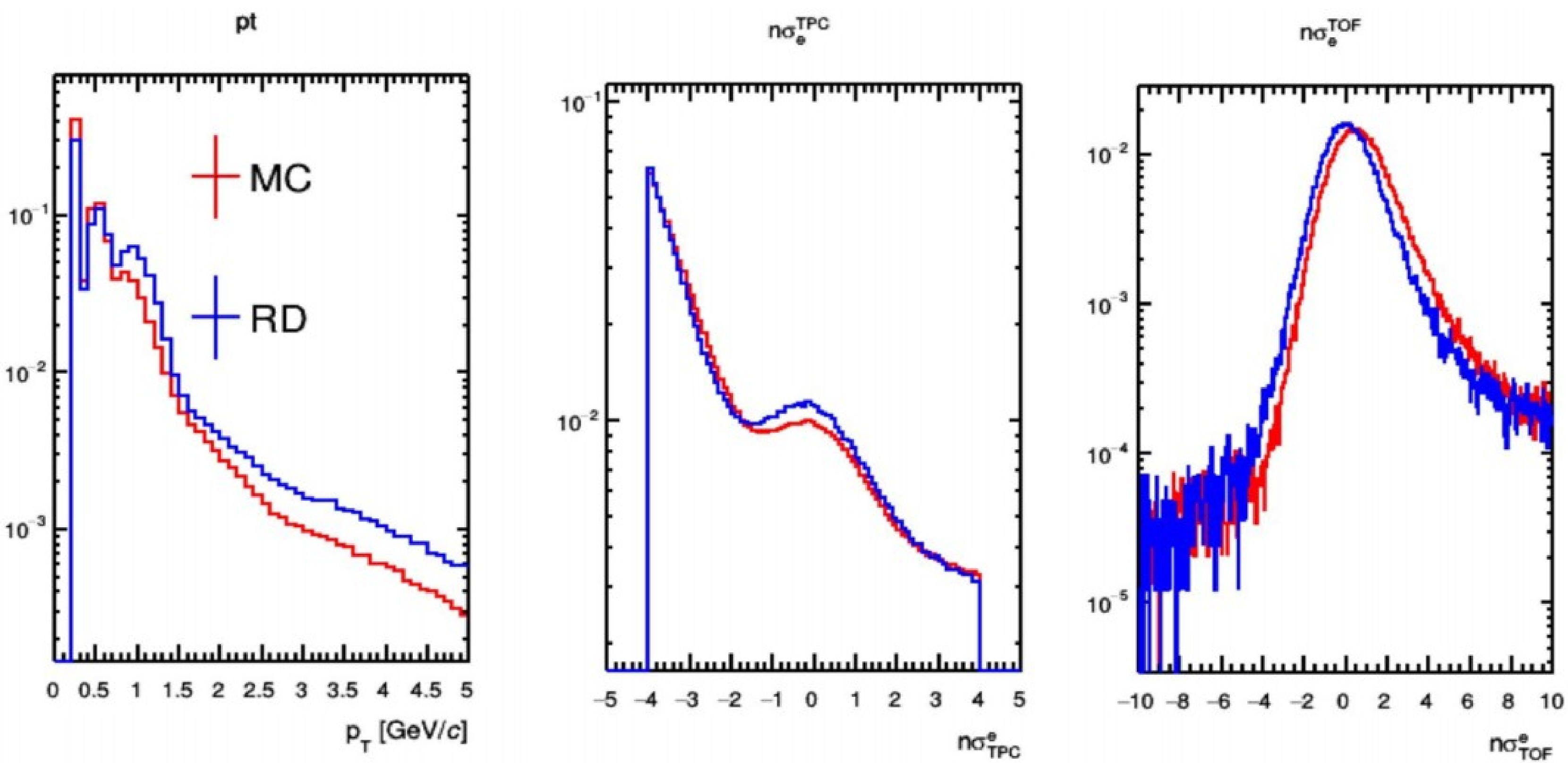
- **Machine learning PID:**
  - perfect task for machine learning
  - can learn non-trivial relations between different track parameters and PID
  - no "trial and error" approach

Proposed solution for PID

- **Build a ML classifier that can outperform traditional PID**
- **Train and validate the classifier on Monte Carlo and real data**
- **Create a simple interface for users (ALICE physicists):**
  - first attempts in 2019 (Random Forest) for LHC Run 2 (AliRoot)
    → proof-of-concept work
  - new, much more advanced, project for LHC Run 3 (O^2)
    → still in research phase
- **Limitations:**
  - Quality of the classifier will depend on the MC sample (need to handle discrepancies between data and MC)
    → no MC reweighting done
  - No easy way to calculate systematic uncertainties from the procedure
  - The classifier is a "black box" - no easy way to tell what's going on inside

Differences MC vs real data

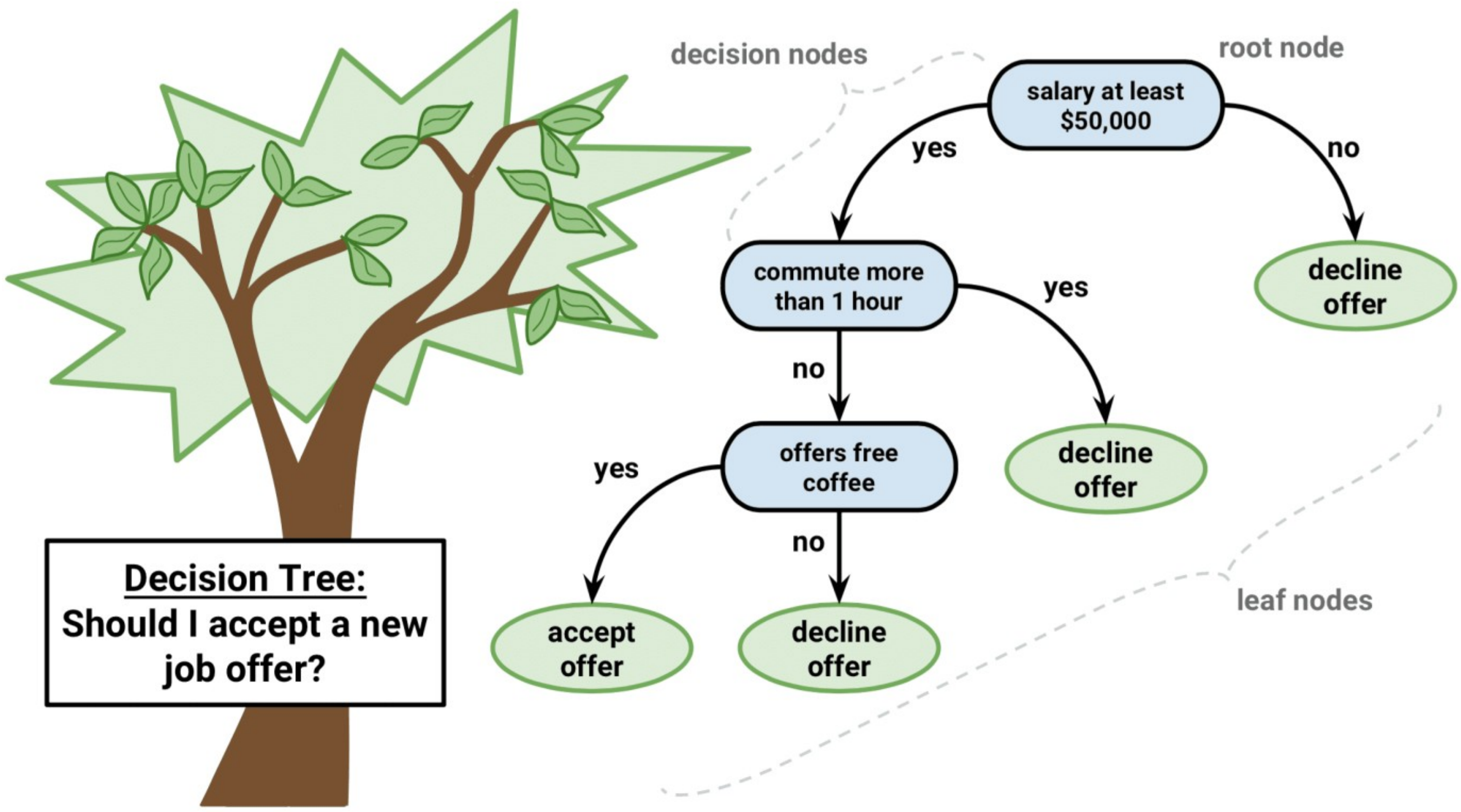
- The MC distributions don't usually reflect real data shapes
- This could potentially have an effect on the quality of identification



LHC Run 2

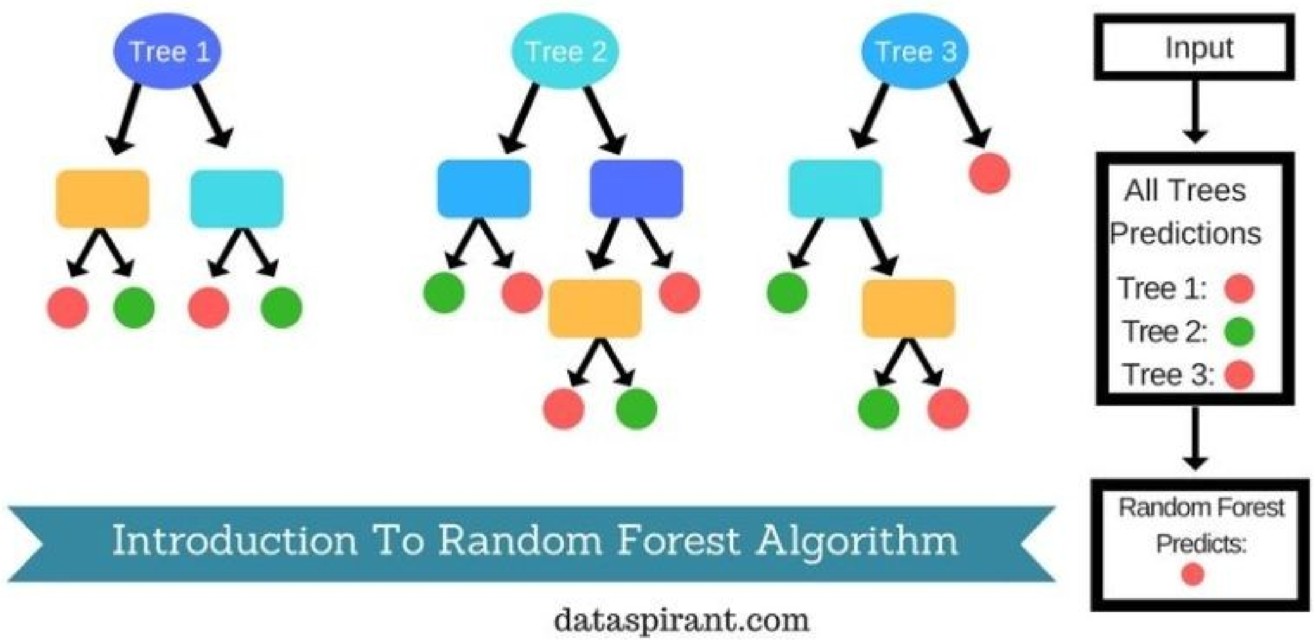
Decision tree

- A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or continues value)
- Decision tree learning uses a decision tree to go from observations about an item (attributes) to conclusions about the item's target value (leaves)



Random Forest

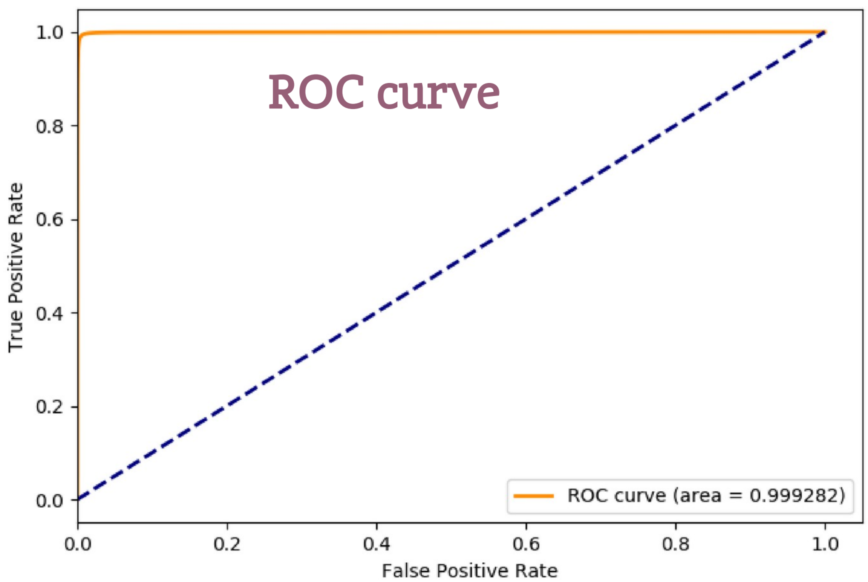
- A collection of decision trees ("forest") where each tree votes for a final decision
- Each tree is trained on a subset of randomly selected training data
- The final result is (in most cases) the one with majority of votes
- ... in addition, adaptive boosting was used

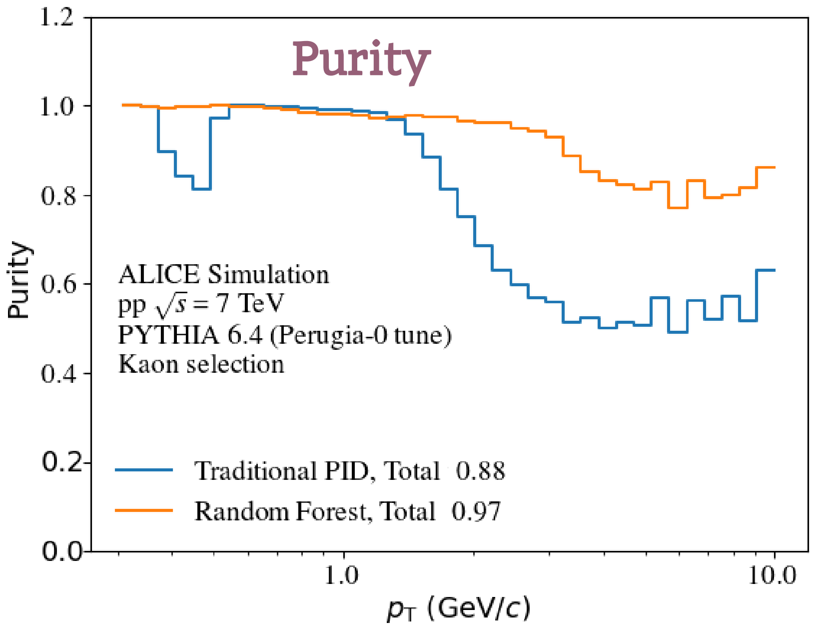


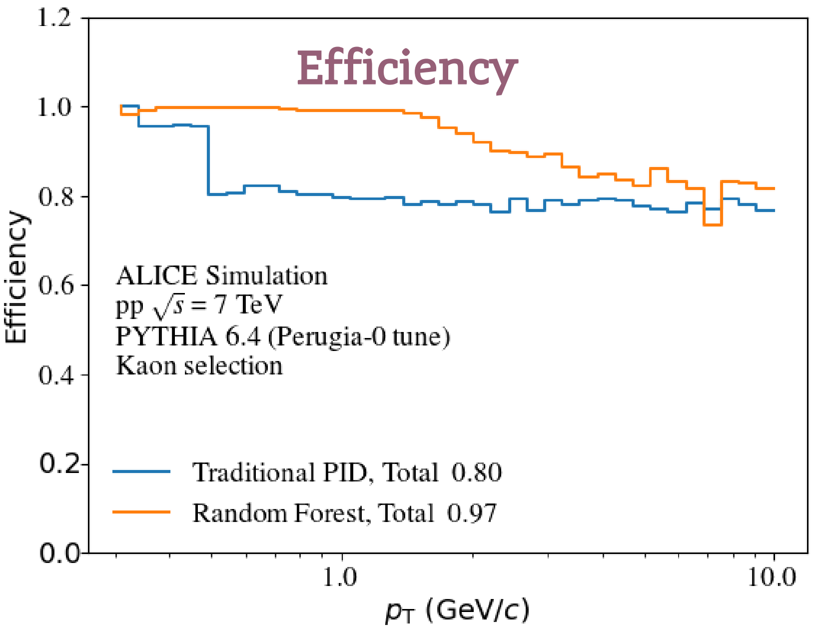
Preliminary results

Results

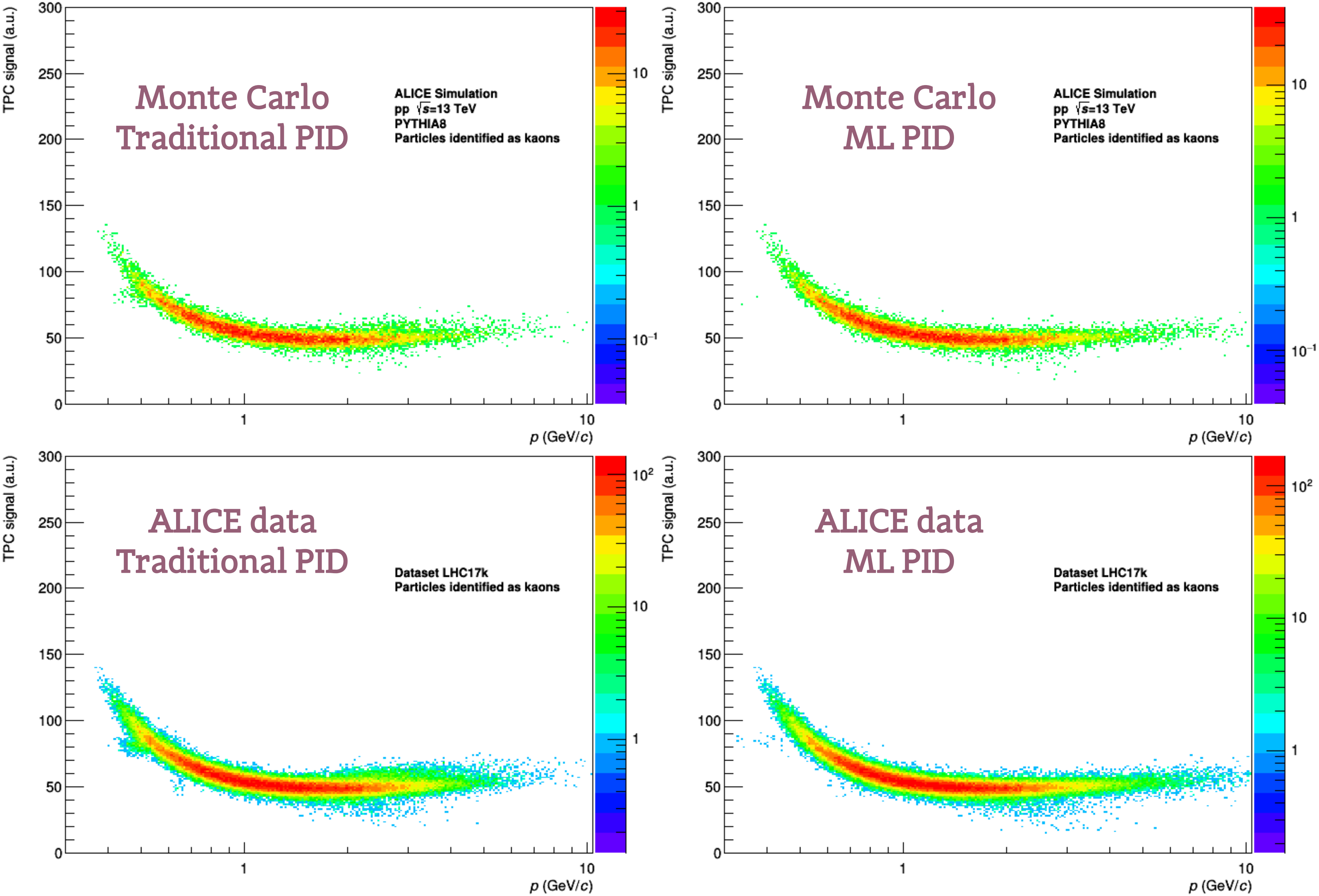
- **Test data sample:**
  - pp @ 7 TeV, Pythia 6 Perugia-0
- **Traditional PID:**
  - $n^2_{\sigma,TPC} < 2$, for $p_T \leq 0.5$ GeV/c
  - $\sqrt{n^2_{\sigma,TPC} + n^2_{\sigma,TOF}} < 2$, for $p_T > 0.5$ GeV/c
- **Machine Learning PID:**
  - Random Forest classifier



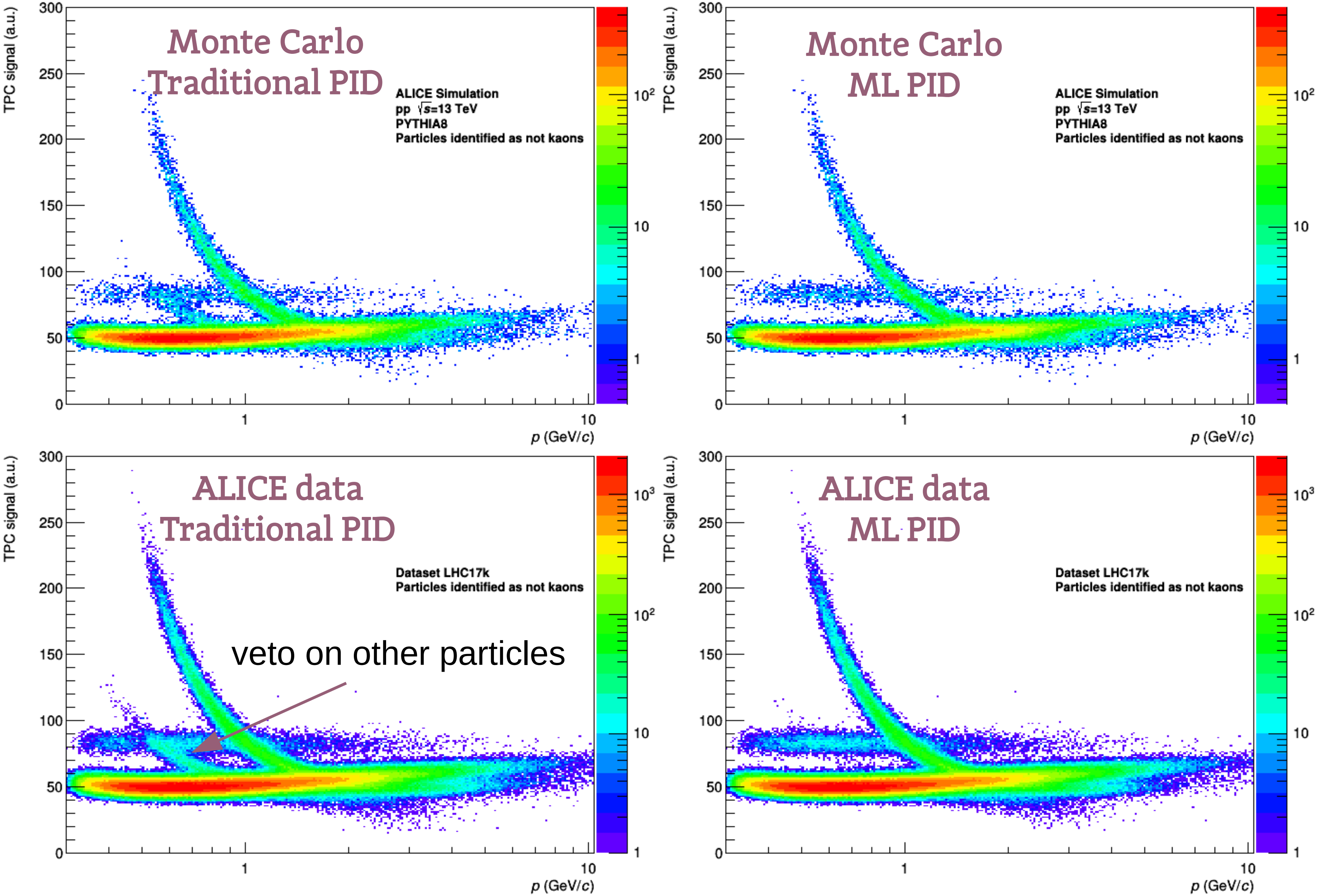




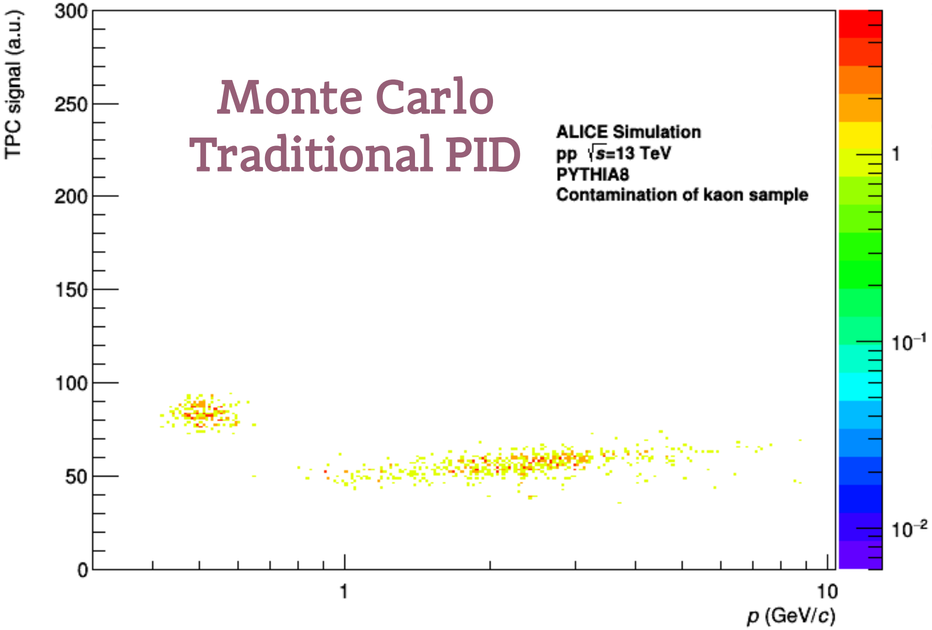
TPC accepted kaons

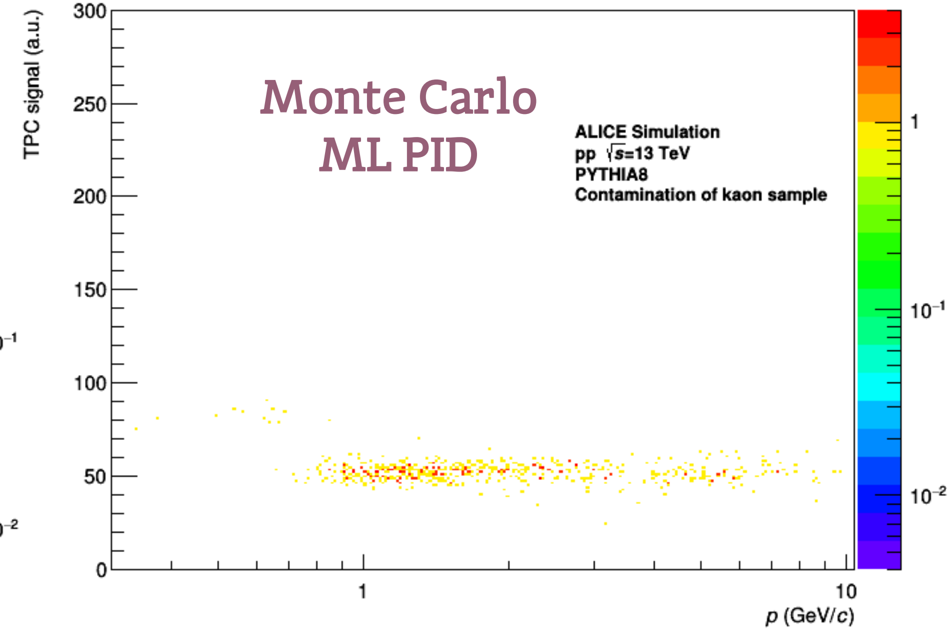


TPC rejected (not kaons)



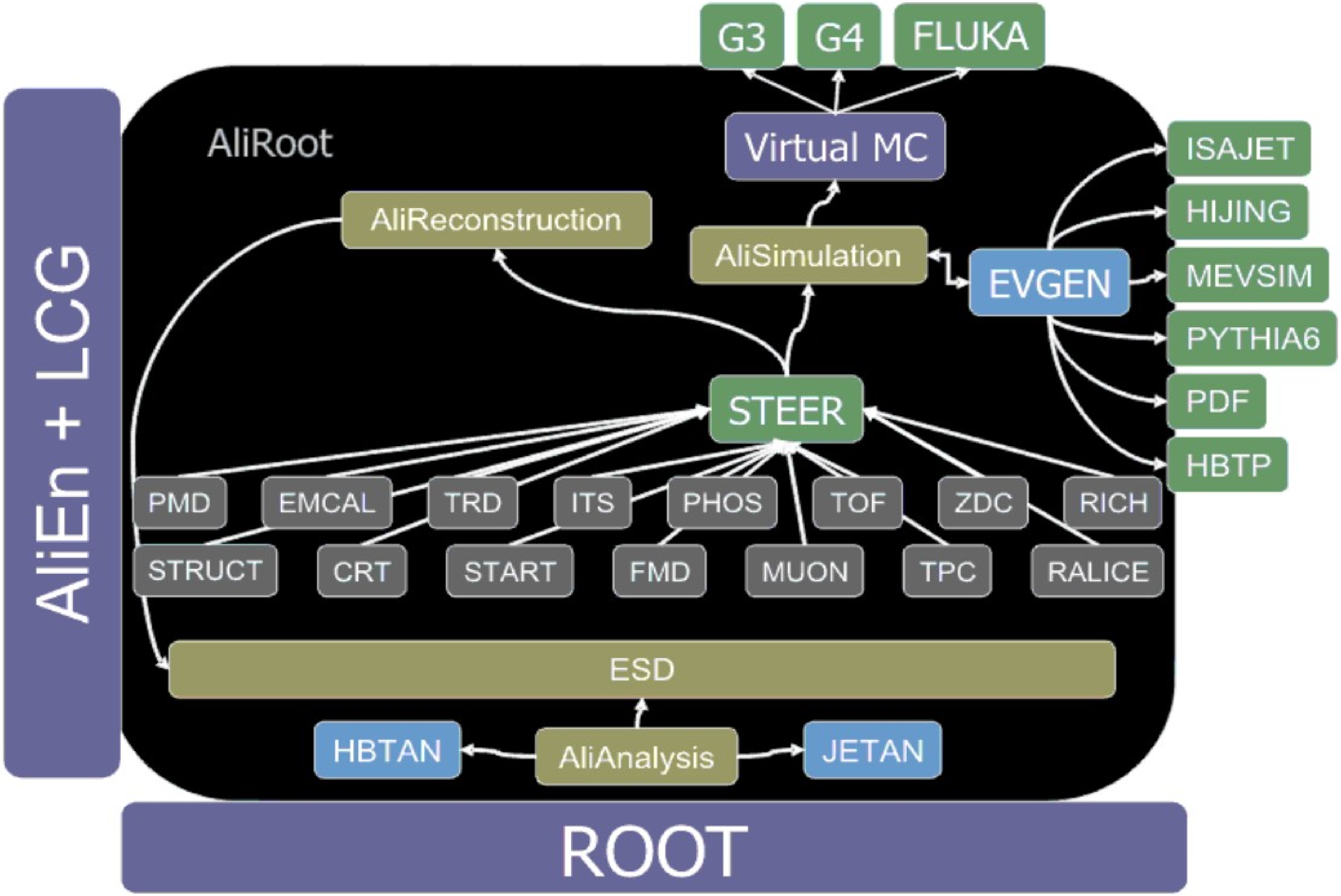
TPC contamination in kaon sample



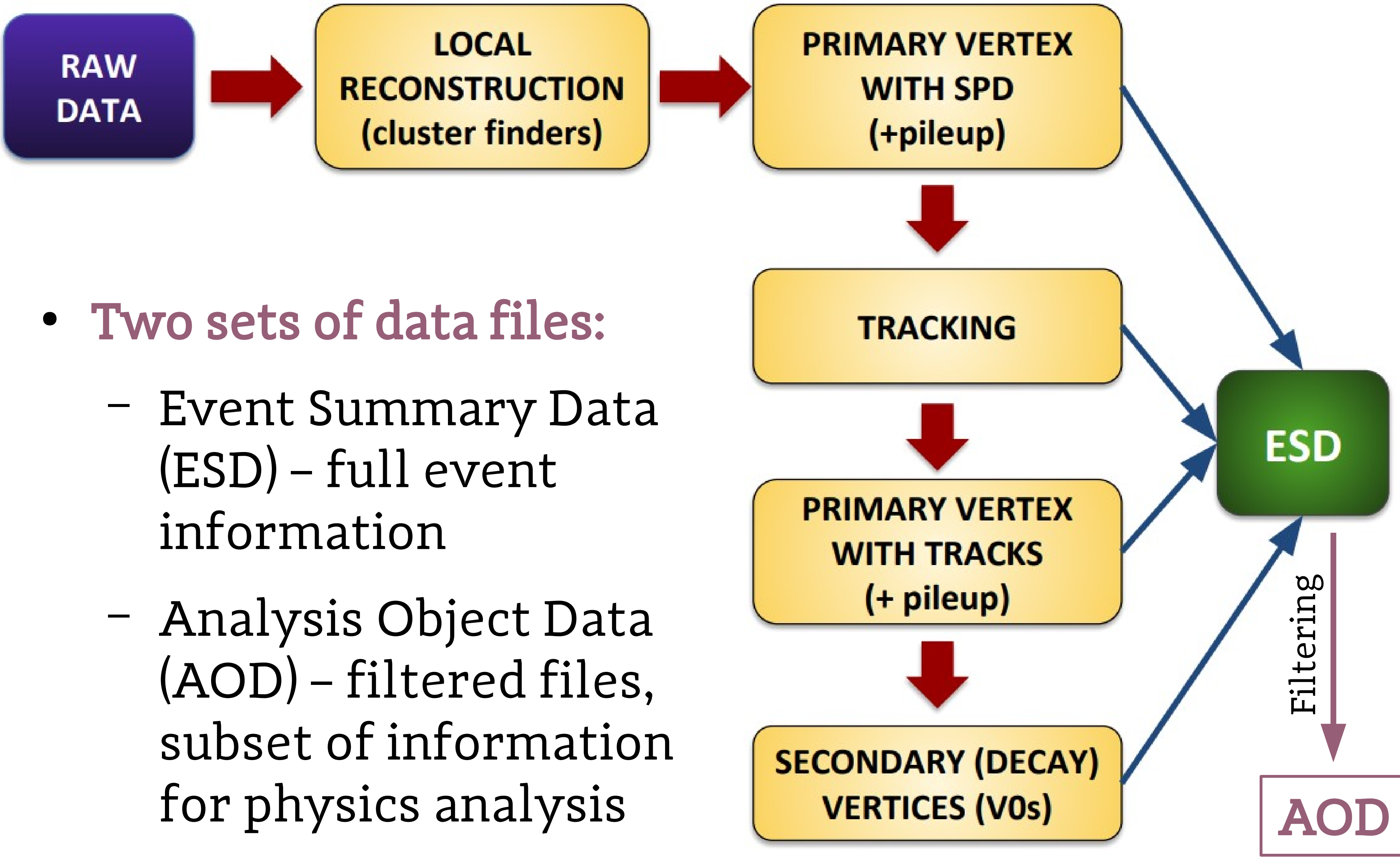


Implementation

ALICE offline framework



ALICE offline framework



- Two sets of data files:
  - Event Summary Data (ESD) – full event information
  - Analysis Object Data (AOD) – filtered files, subset of information for physics analysis

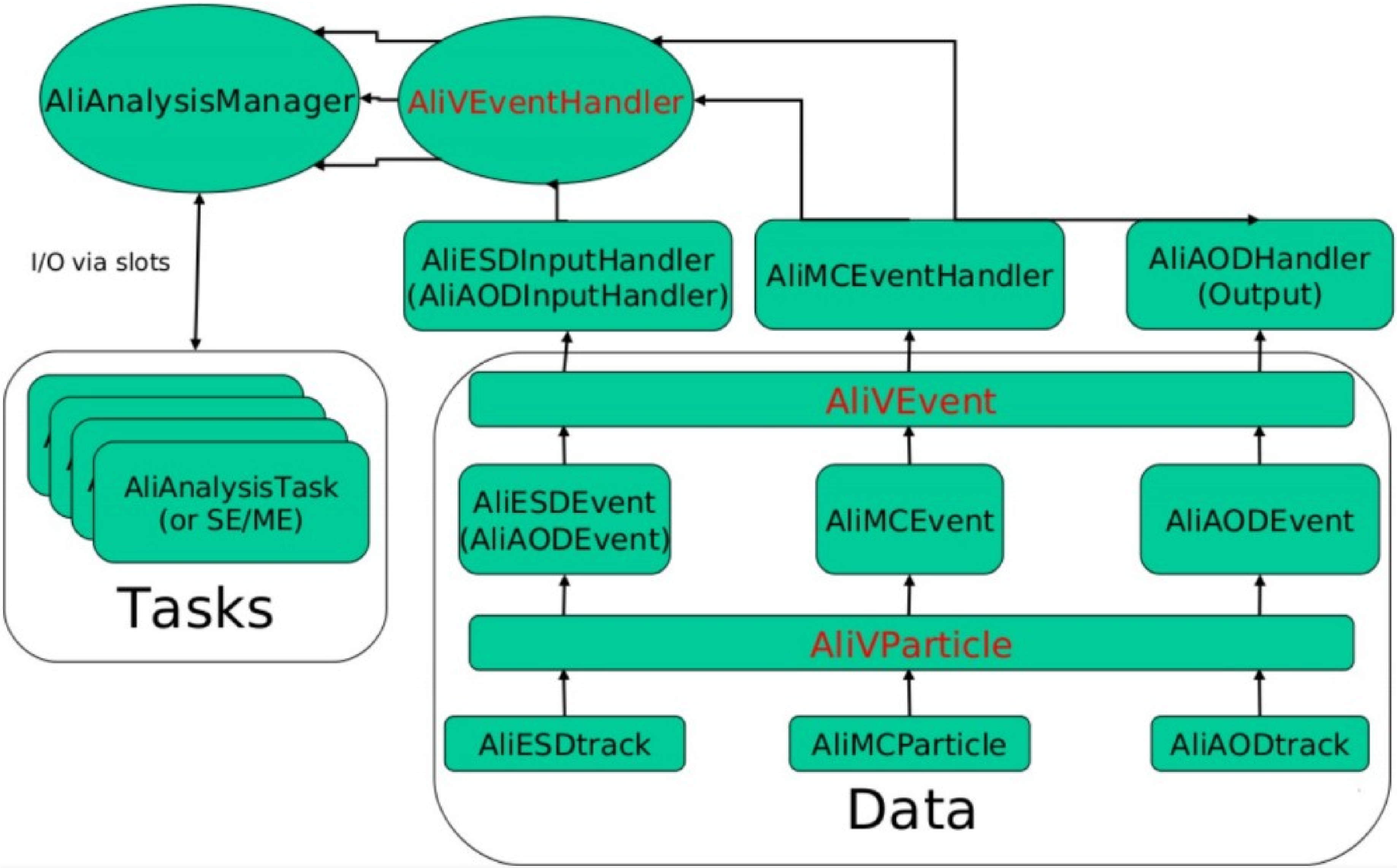
User Tasks

- Analysis is performed in an automatized way by the framework
- Users write their analysis tasks, which are specific C++ classes in AliRoot
- Framework provides iterations over files and events

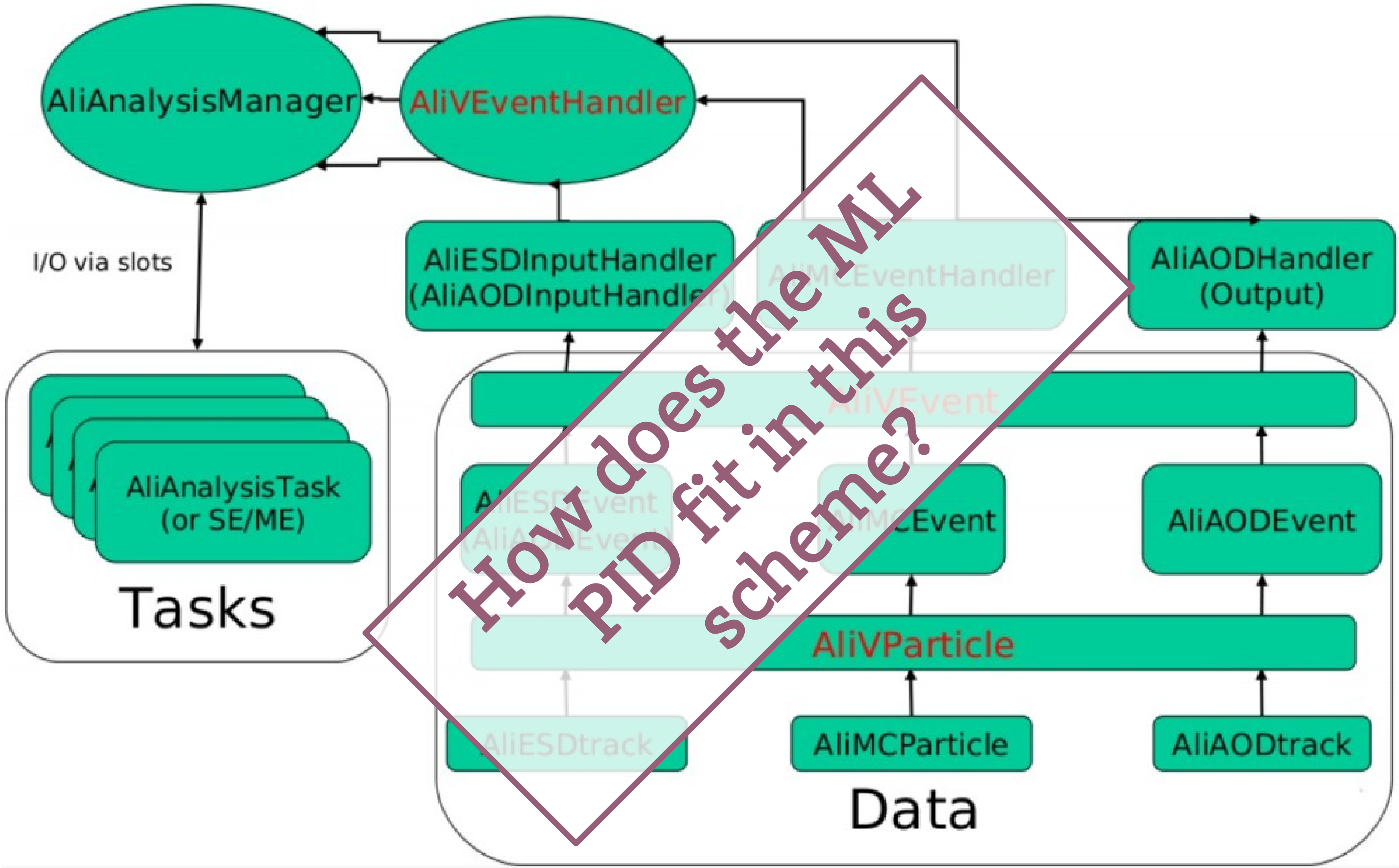
- "Constructor“* — called once on local PC
- UserCreateOutputObjects()
- UserExec() — for each event
- Terminate()

*Called in the macro

AliRoot analysis scheme



AliRoot analysis scheme

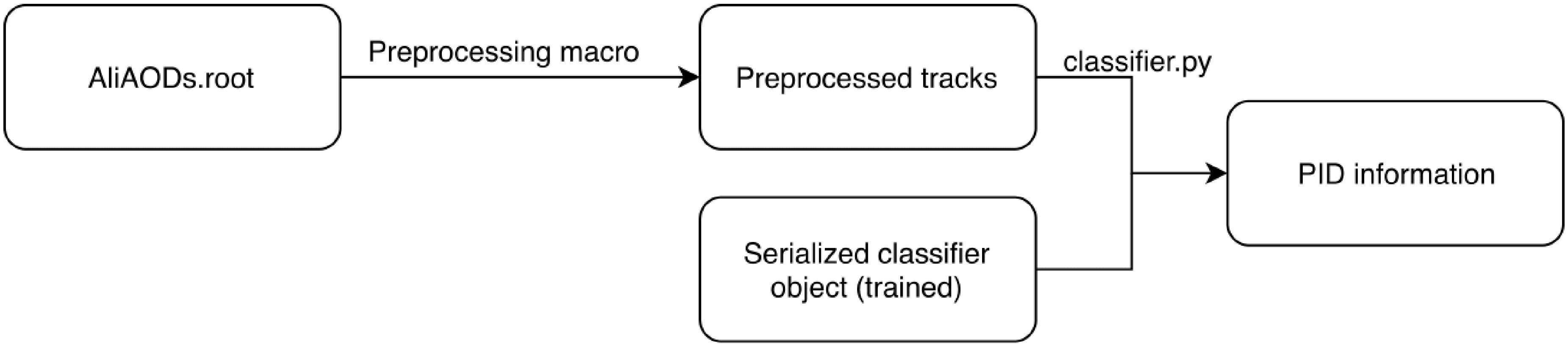


Implementation attemps

- **Training part**
  - Not covered in this talk, done externally to ALICE software
  - Proposed solution: to be done in a centralized way → not implemented finally in AliRoot
- **Classification part**
  - Classifier (in Python) prepared by an IT student
  - Implementation work in AliRoot by a physics student
  - Different attempts tested based on framework limitations
  - Demo/beta version prepared

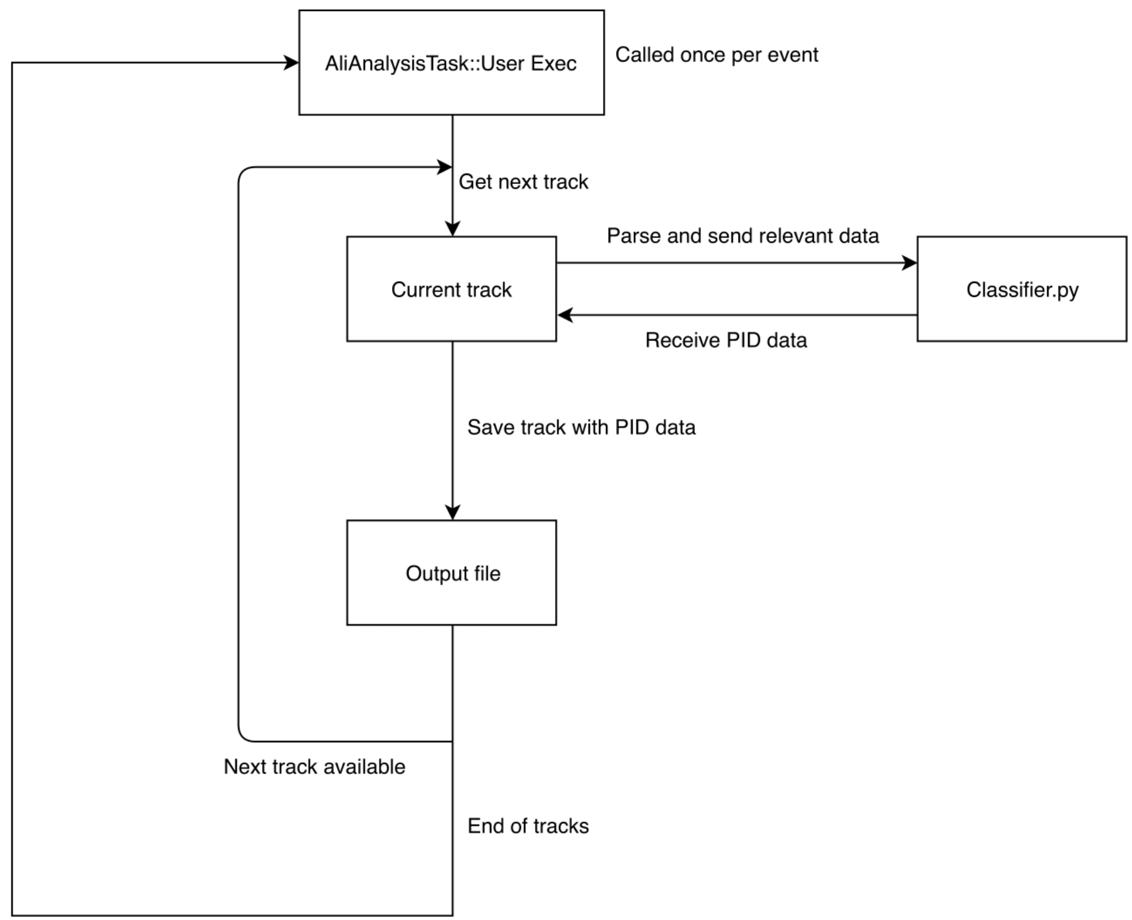
Classification – general idea

- Take tracks from AOD files and the trained model (classifier.py) in Python
- Propagate AOD tracks through the model to get the ML information for each track
- The ML PID information consists of predicted probabilities for PDG codes (pion, kaon, proton, electron, muon)
- Present the information to the user
  - via specific objects accessible in AliRoot



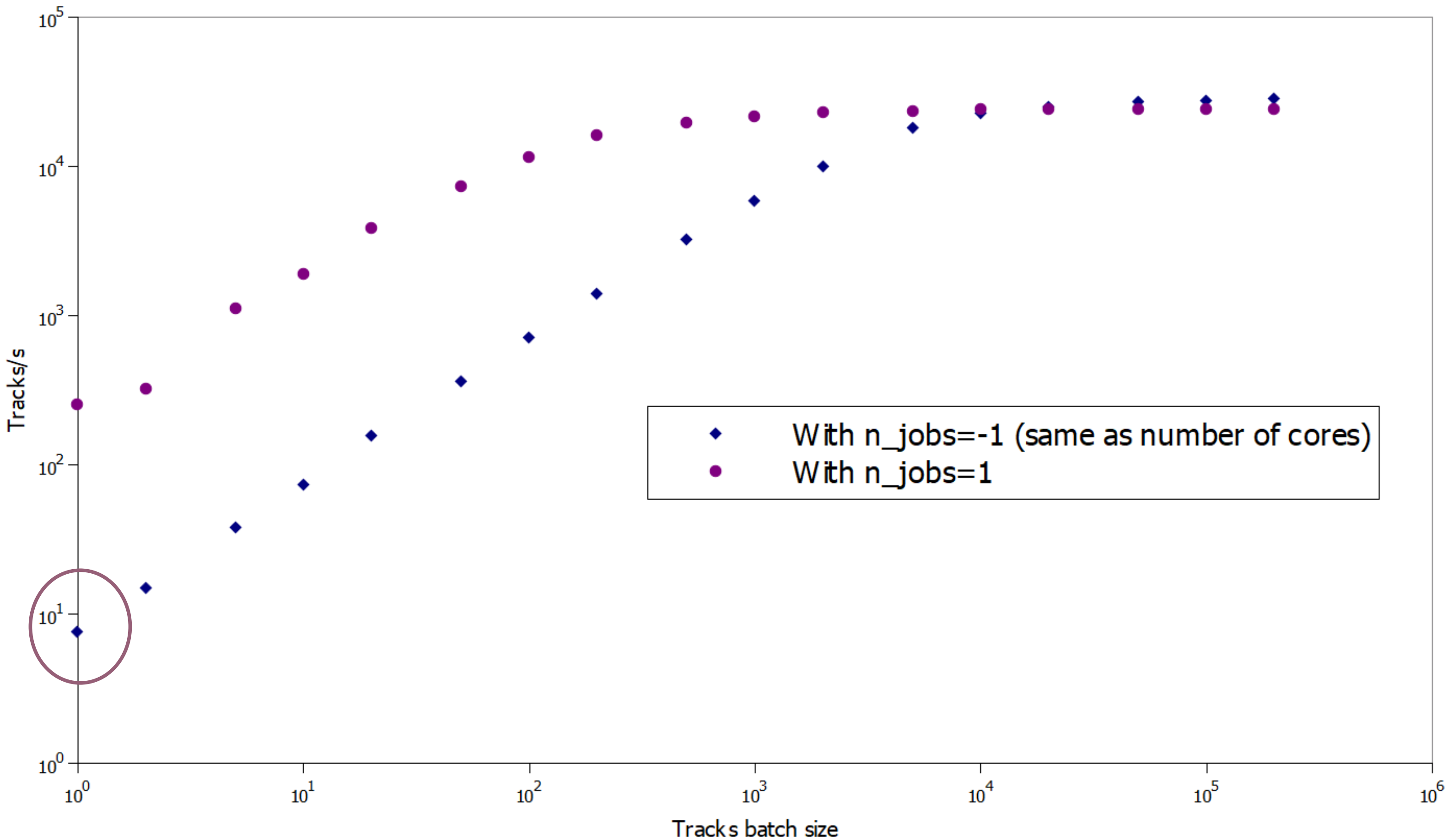
First attempt

- Track-by-track implementation
- Framework to iterate over events, loop over tracks in `UserExec()`
- Classifier listens in the background
- Stripped files sent via pipe
- PID results received via another pipe
- The method is VERY SLOW



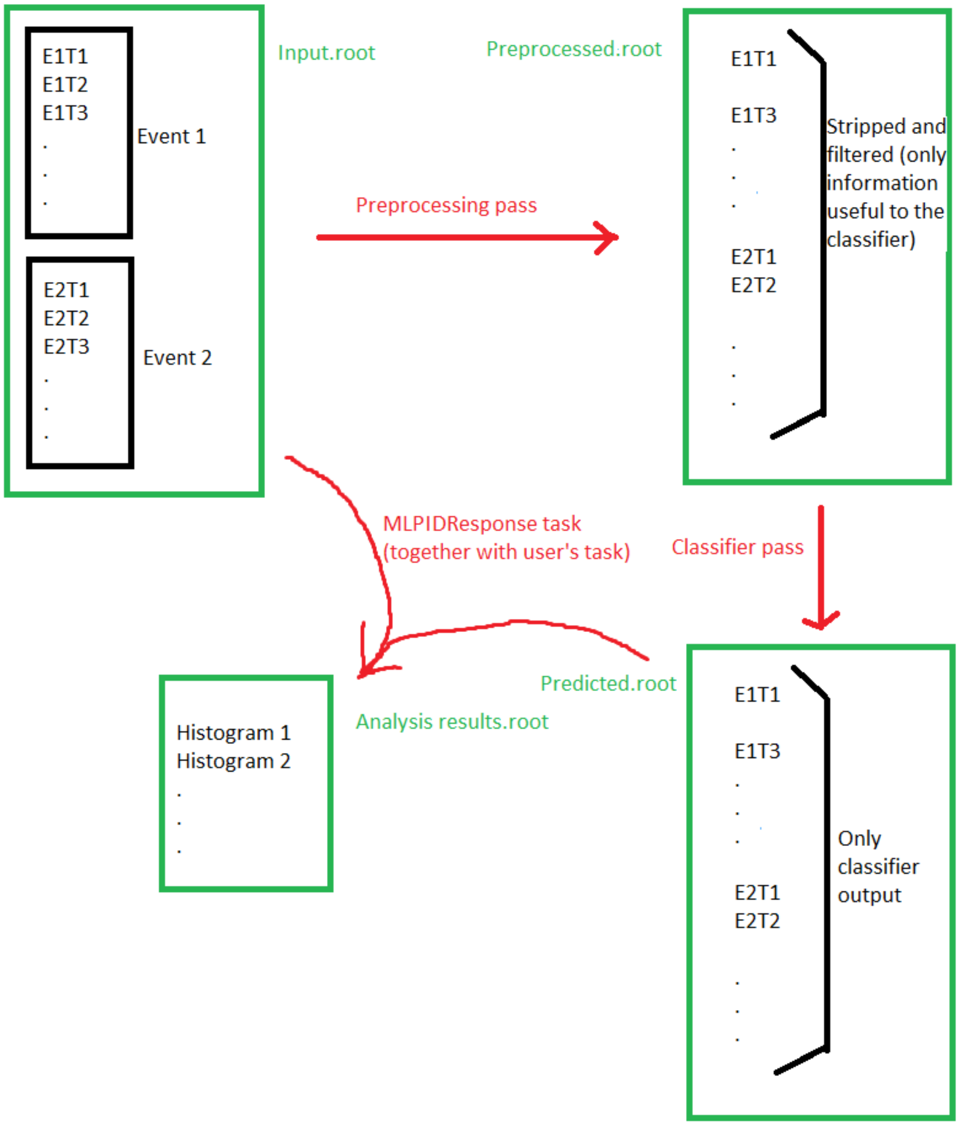
Scikit-learn benchmark

- In default track-by-track implementation, with threads, we can process <u>only ~9 tracks/s</u> (overhead from the thread creation) → no multiple threads allowed on the GRID
- Increase to more than 100 tracks/s if we do not allow threads



Second attempt

- Propagate multiple tracks through the classifier
- Two loops over events needed
  - create a temporary (stripped) file
  - propagate the temporary file through the classifier
  - produce **predicted.root** file
- In the second loop over events use a lookup table to match the two files
- Solution in AliRoot difficult (processing events twice), also slower than regular analysis



Final attempt

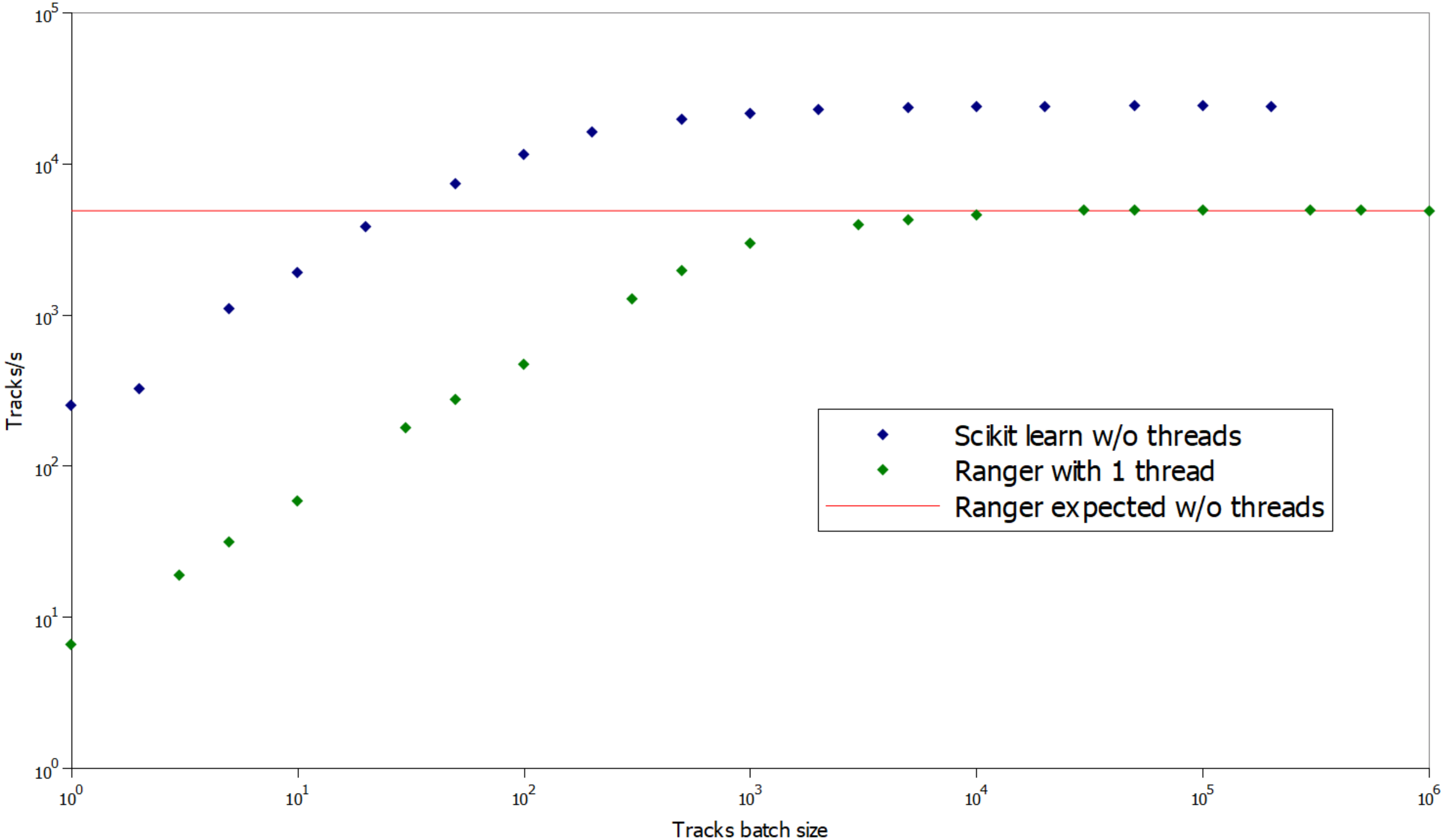
- Propagate multiple tracks through the classifier combined from <u>single events</u> (do not combine multiple events)
  - computational time of a simple p_T analysis task with ML PID (scikit-learn) and without ML PID (one 200 MB AOD file):

Real time 0:00:34 --- Without ML PID

Real time 0:01:33 --- With ML PID

  - the analysis with ML PID is **3x slower** than without ML PID
- Python interface not easily available in AliRoot, use the C++ Random Forest library (for example Ranger) instead of Python
- First tests:
  - created a "random" C++ Random Forest of the same size and depth
  - compare Ranger and scikit-learn speed tests (next slide)

Scikit-learn vs Ranger



- Ranger (C++) is <u>slower</u> than scikit-learn (Python) → Python is faster
- Ranger creates threads even when set to 1

Working demo/beta example

run macro

```
AliAnalysisTaskMLPt *myTask = new AliAnalysisTaskMLPt("MyTask");
AliAnalysisTaskMLPIDResponse *mlpidTask = new AliAnalysisTaskMLPIDResponse("MLPIDTask");

myTask->SelectCollisionCandidates(AliVEvent::kINT7);
if( !myTask )
  exit(-1);
mgr->AddTask(mlpidTask);
mgr->AddTask(myTask);

// Create containers for input/output
AliAnalysisDataContainer *cinput = mgr->GetCommonInputContainer();
AliAnalysisDataContainer *coutput2 = mgr->CreateContainer("MyTree",
        TList::Class(),AliAnalysisManager::kOutputContainer,outfilename);

//connect them to future analysis
mgr->ConnectInput(mlpidTask,0,cinput);
//mgr->ConnectOutput(mlpidTask,1,coutput2);
mgr->ConnectInput(myTask,0,cinput);
mgr->ConnectOutput(myTask,1,coutput2);

if ( !mgr->InitAnalysis() )
    return;
mgr->PrintStatus();

//start analysis
mgr->StartAnalysis("local",chain,Nevents);
```

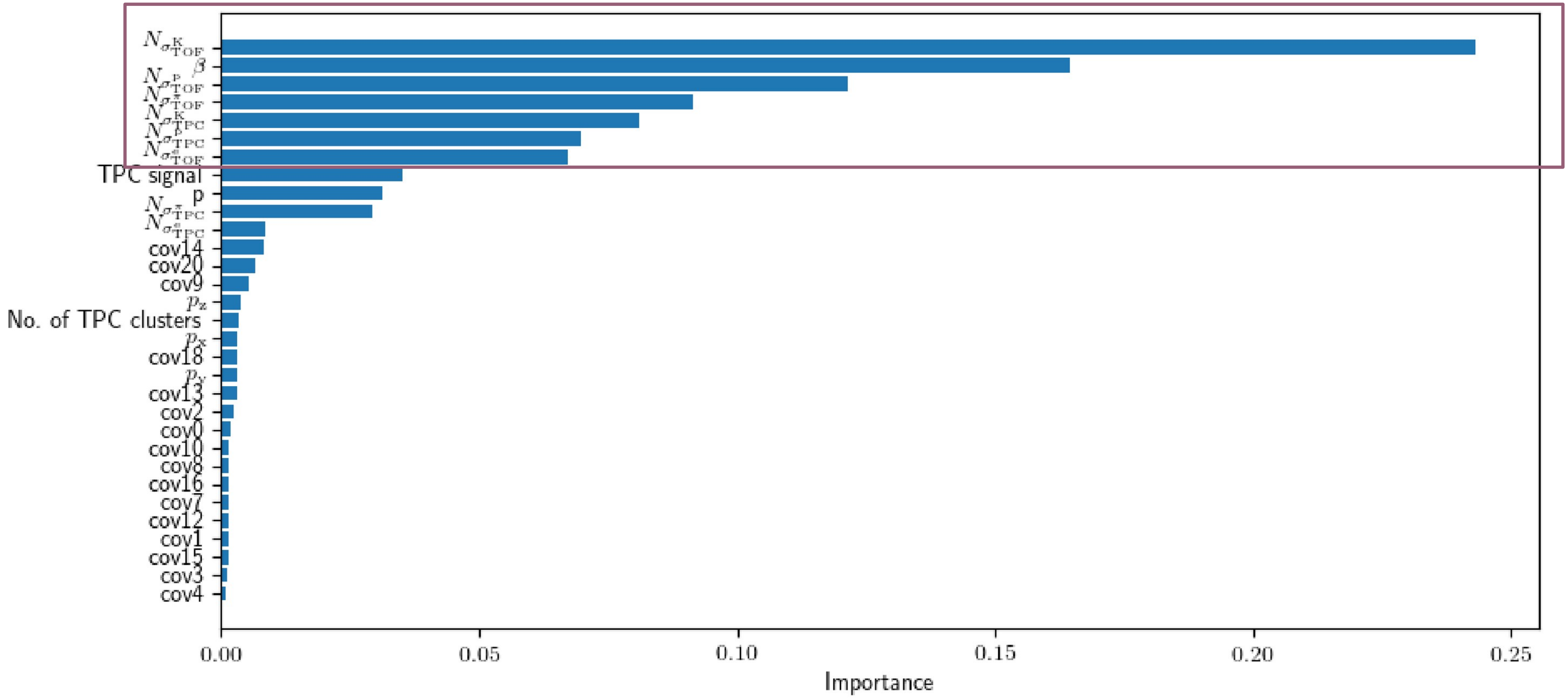
user's analysis task

```
//loop over AOD reconstructed tracks
for (Int_t iTracks = 0; iTracks < aodEvent->GetNumberOfTracks(); iTracks++) {
    //get track
    AliAODTrack *track = (AliAODTrack*)aodEvent->GetTrack(iTracks);
    if (!track)
        continue;

    UInt_t filterBit = 96;
    if(!track->TestFilterBit(filterBit))
        continue;

if (!fMLpidUtil)
  continue;

AliMLPIDResponse* resp = fMLpidUtil->getTrackPIDResponse(track->GetID());
if (!resp)
  continue;
else
  cout<<"Good PID: "<<resp->predictedPDG<<endl;


int pdg = resp->predictedPDG;
if(pdg == 211)
  ptHistPions->Fill(track->Pt());
if(pdg == 321)
  ptHistKaons->Fill(track->Pt());
if(pdg == 2212)
  ptHistProtons->Fill(track->Pt());


    //save all attributes into TTree
    //treeOutput->Fill();
}
```

- **User just needs to add a couple of lines – like for a traditional PID**
  → inclusion of the PID response task

LHC Run 3

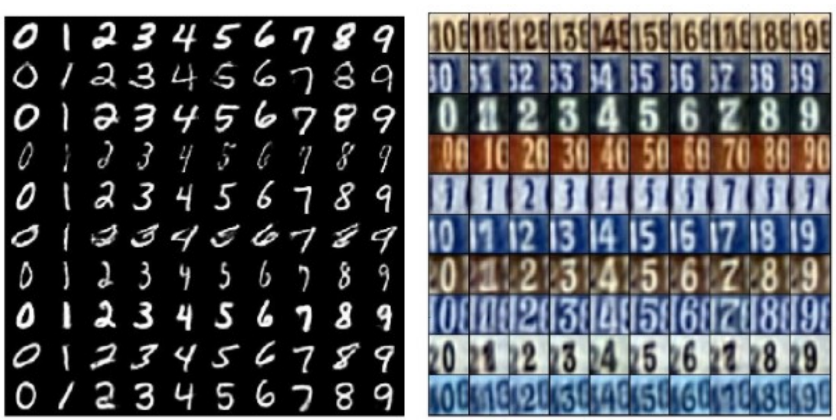
PID parameter importance



- The algorithm was trained on information from TPC and TOF parameterization (which is done before and loaded with "PID response task" in the analysis)
- ... in the LHC Run 3 we plan to use only raw signals from the detectors (TPC dE/dx, TOF time)

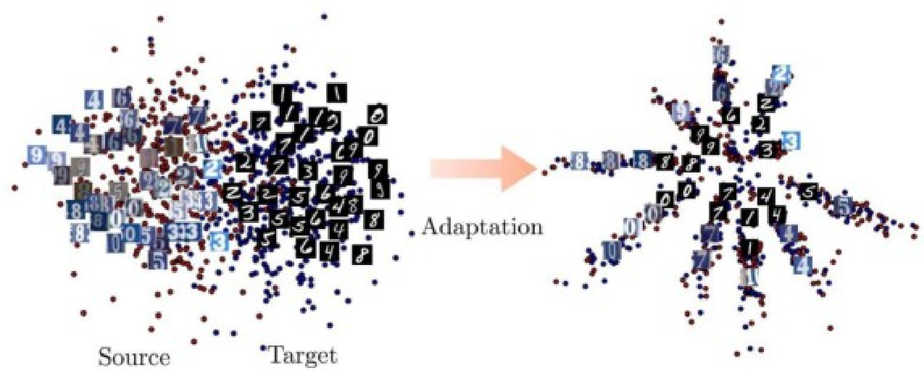
Domain adaptation

- ALICE is undergoing a major upgrade with completely new software framework O^2
- We plan to explore the Unsupervised Domain Adaptation for ML PID
  - problem of transferring the knowledge from a **labeled source domain** to **unlabeled target domain**, when both domains have different distributions of attributes (as in the case of MC and data)
- No implementation in O^2 yet, research work ongoing



(a) MNIST (b) SVHN

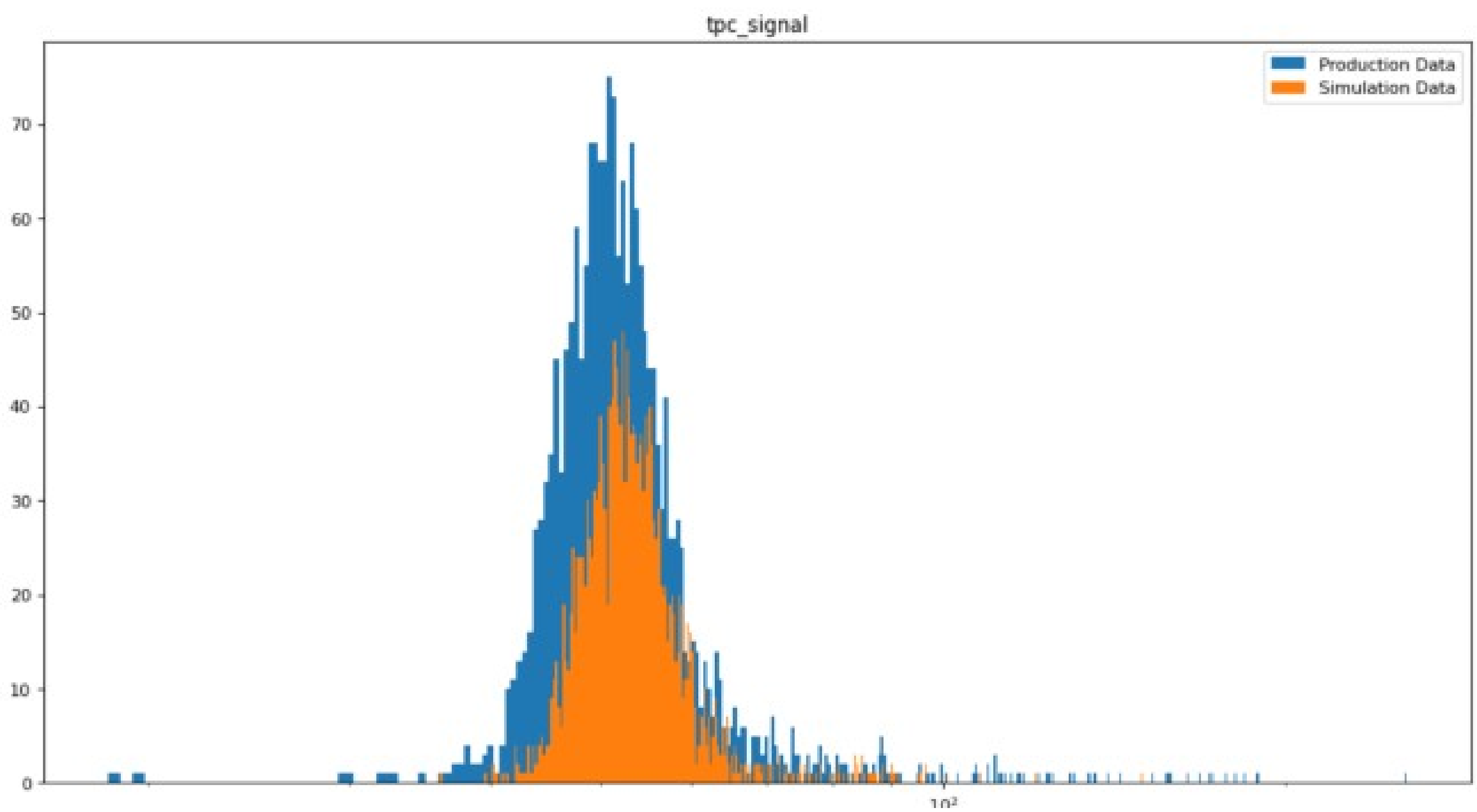
MNIST and SVHN datasets



Visualization of domain adaptation

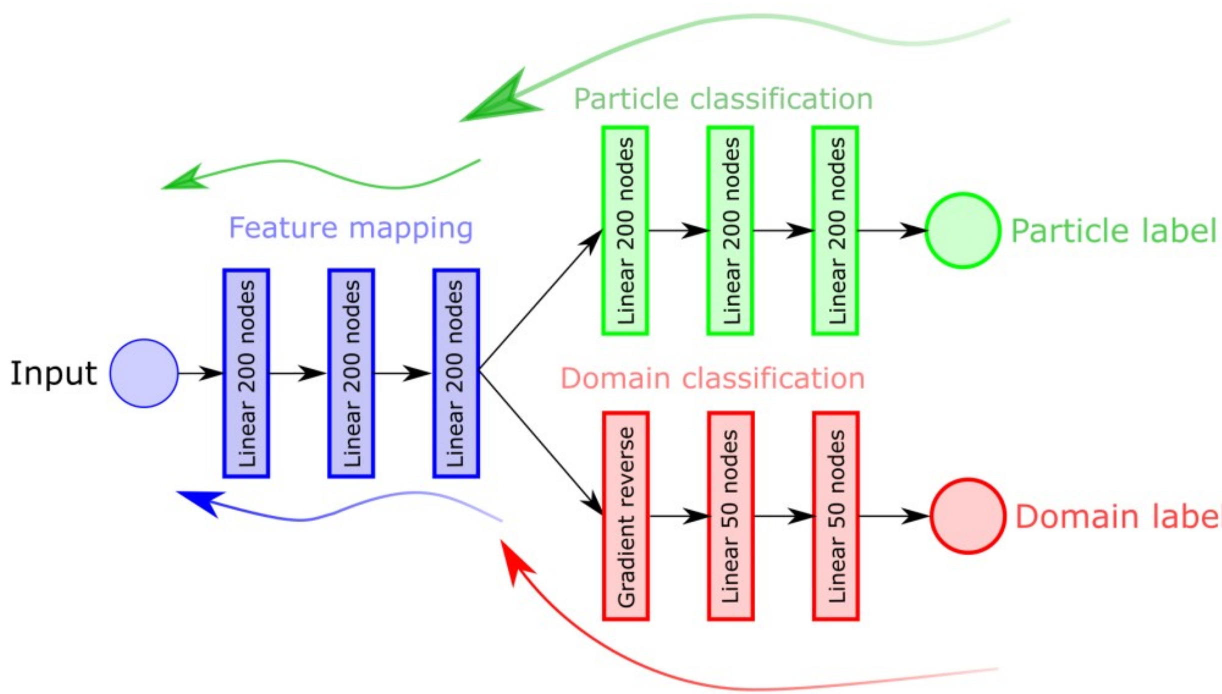
Domain adaptation

- Example domain shift between MC simulated and real data (TPC signal)

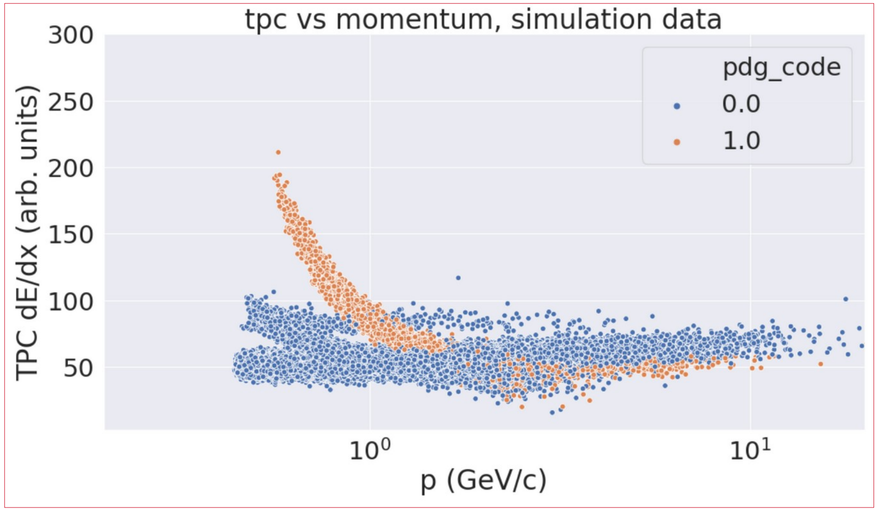


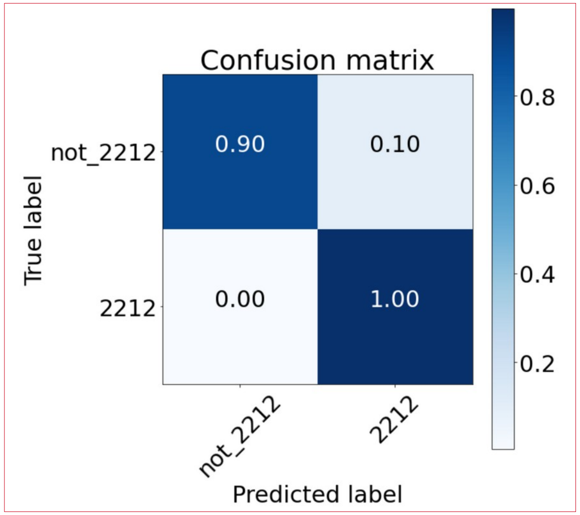
Proposed model

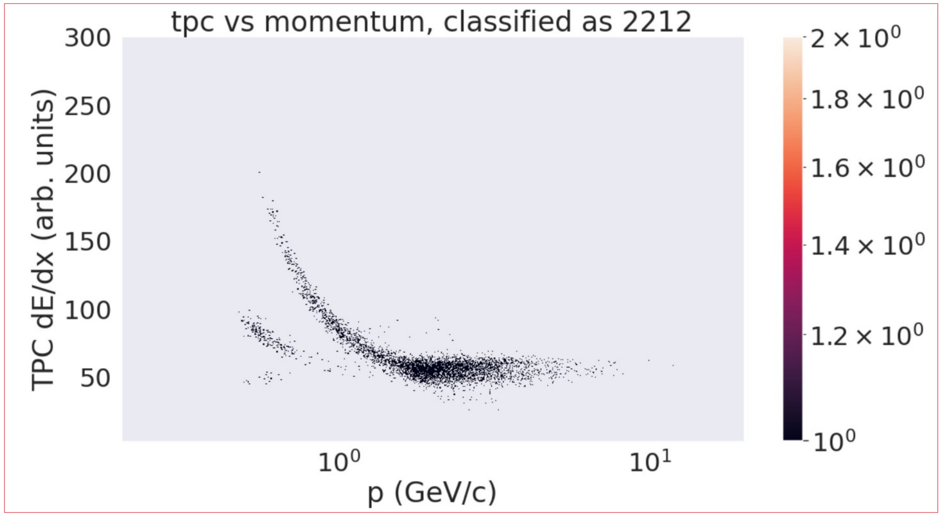
- One versus all based model based on Domain Adversarial Training of Neural Networks
- Architecture consists of three neural networks:
  - **feature mapping network**, which maps features of both data sets into common, domain invariant latent space
  - **particle classification network**, which classifies particles basing on domain invariant latent space
  - **domain discriminator network**, which classifies domain of each particle



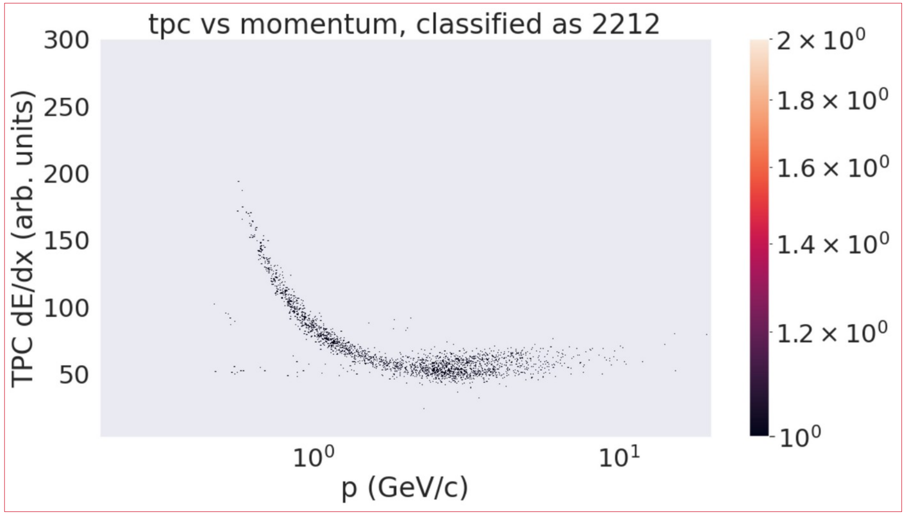
First results – proton selection







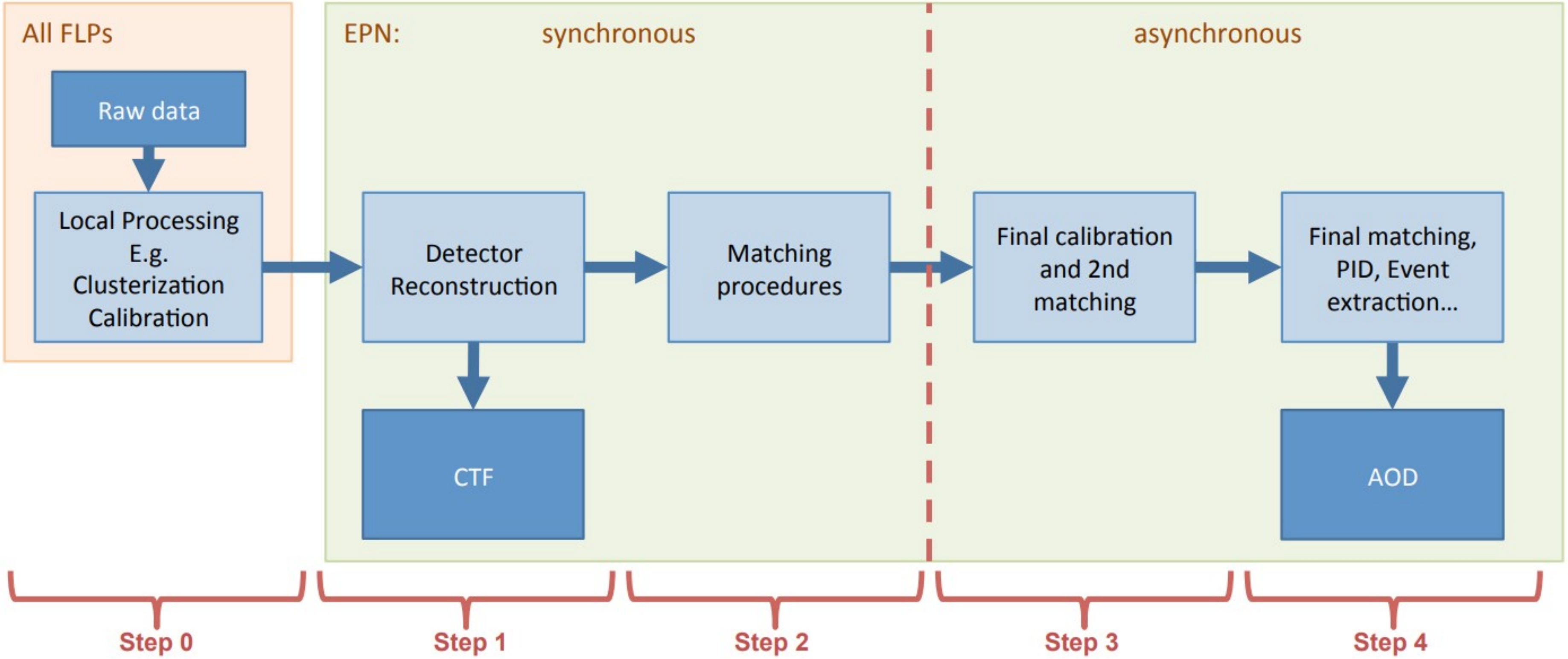
No Domain Adaptation



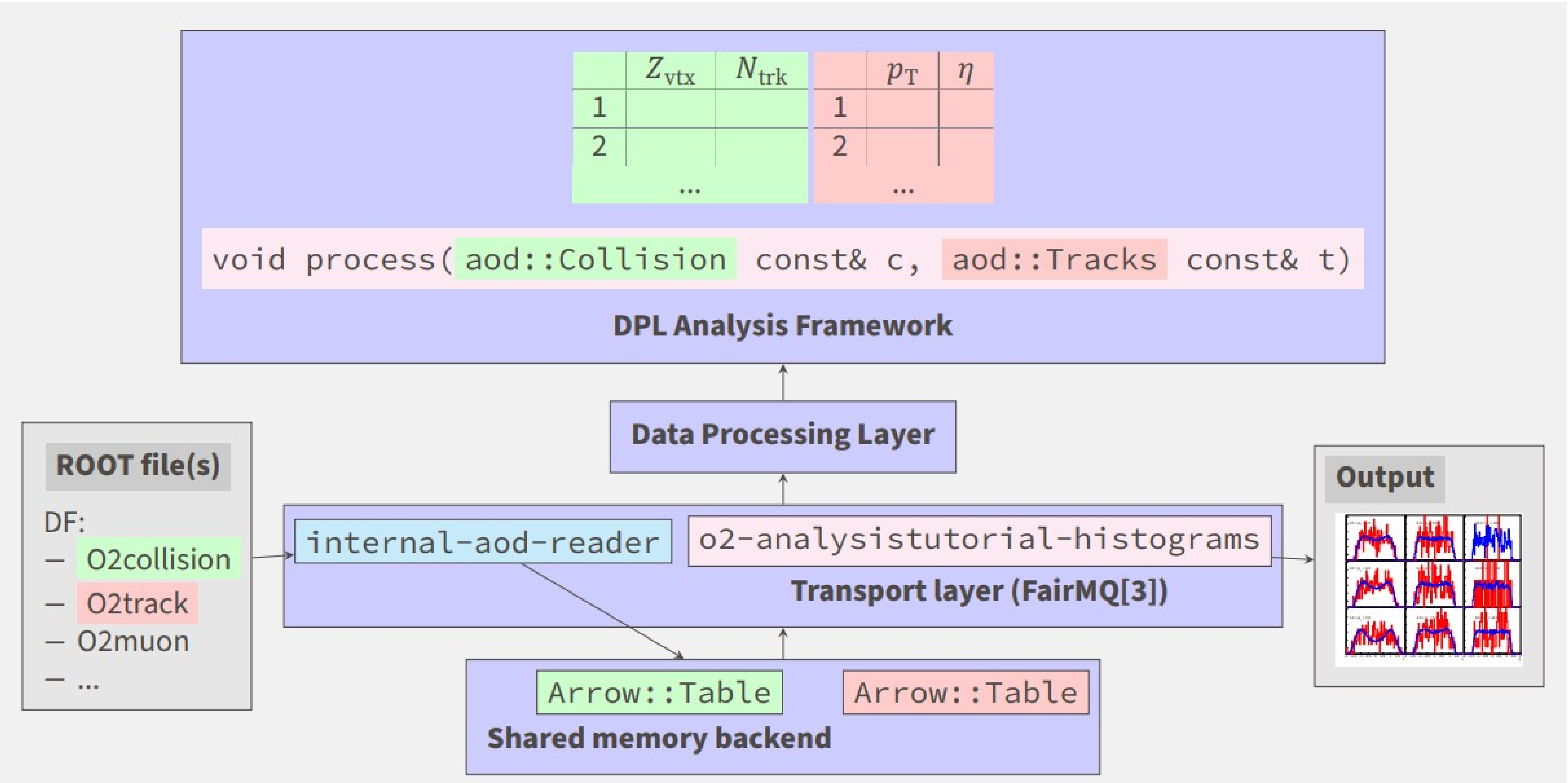
Domain Adaptation

LHC Run 3 computing (O^2)

- x100 higher data rate
- (Updated) AOD format – calculate as much as possible on-the-fly
- O2 Data Processing Layer (DPL)
  - coherent framework from data taking to analysis
- Input data - flat tables (sets of columns) stored as flat ROOT trees



LHC Run 3 computing (O^2)



Anton Alkin, vCHEP 2021
https://indico.cern.ch/event/948465/contributions/4324158/

LHC Run 3 computing (O^2)

- Example analysis task

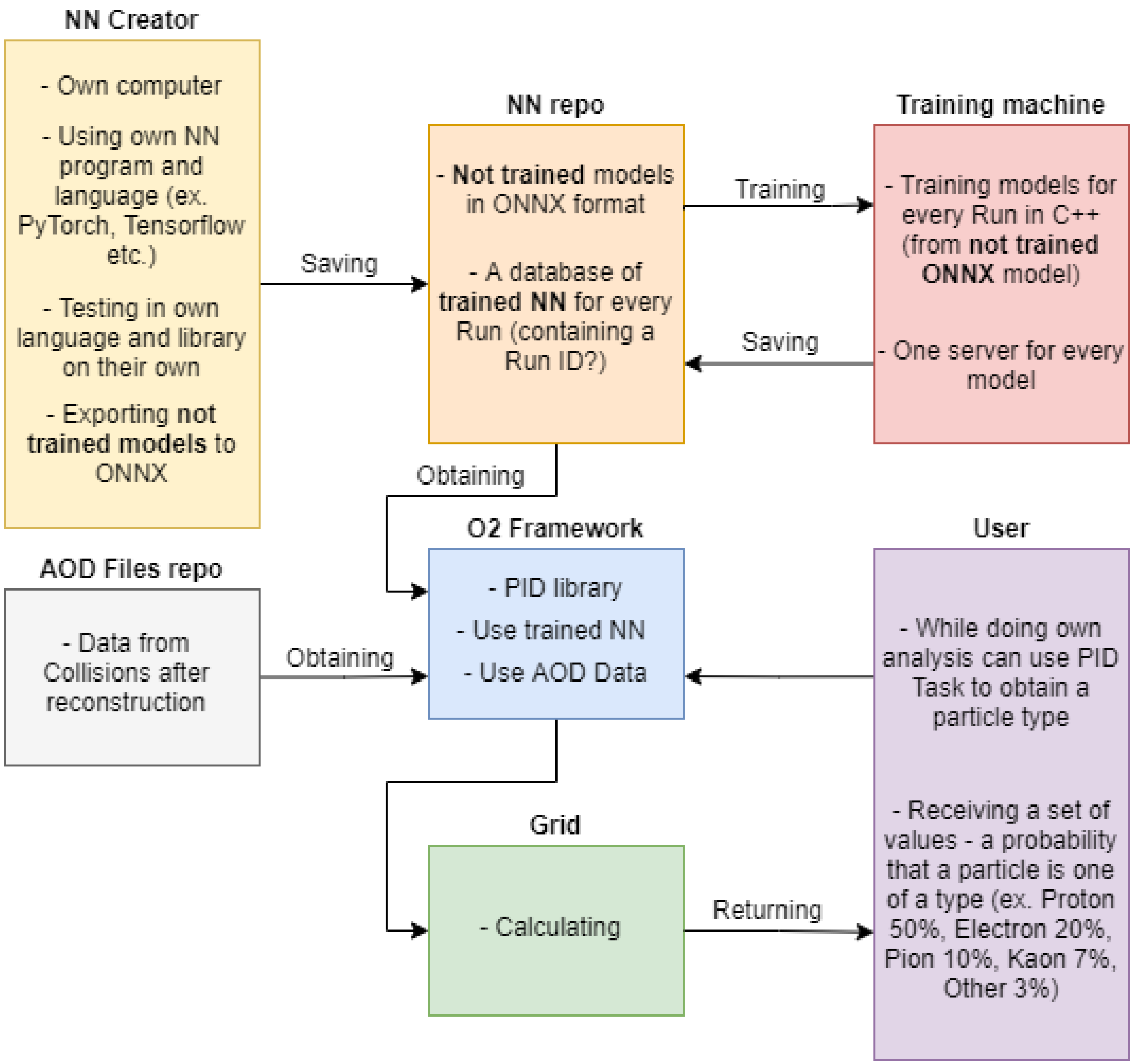
```
struct ATask {
  OutputObj<TH2F> etaphiH{TH2F("etaphi", "etaphi", 100, 0., 2. * M_PI,
102, -2.01, 2.01)};
  Configurable<float> phiCut{"phiCut", 6.29f, "A cut on phi"};
  Filter phiFilter = (track::phi < phiCut)

  void process(soa::Filtered<Tracks> const& tracks)
  {
    for (auto& track : tracks) {
      etaphiH->Fill(track.phi(), track.eta());
    }
  }
};
```

Only filtered tracks

- Our ML PID model has to fit in this scheme!

Implementation

- ONNX discussed to be used for storing trained networks
- Very preliminary scheme

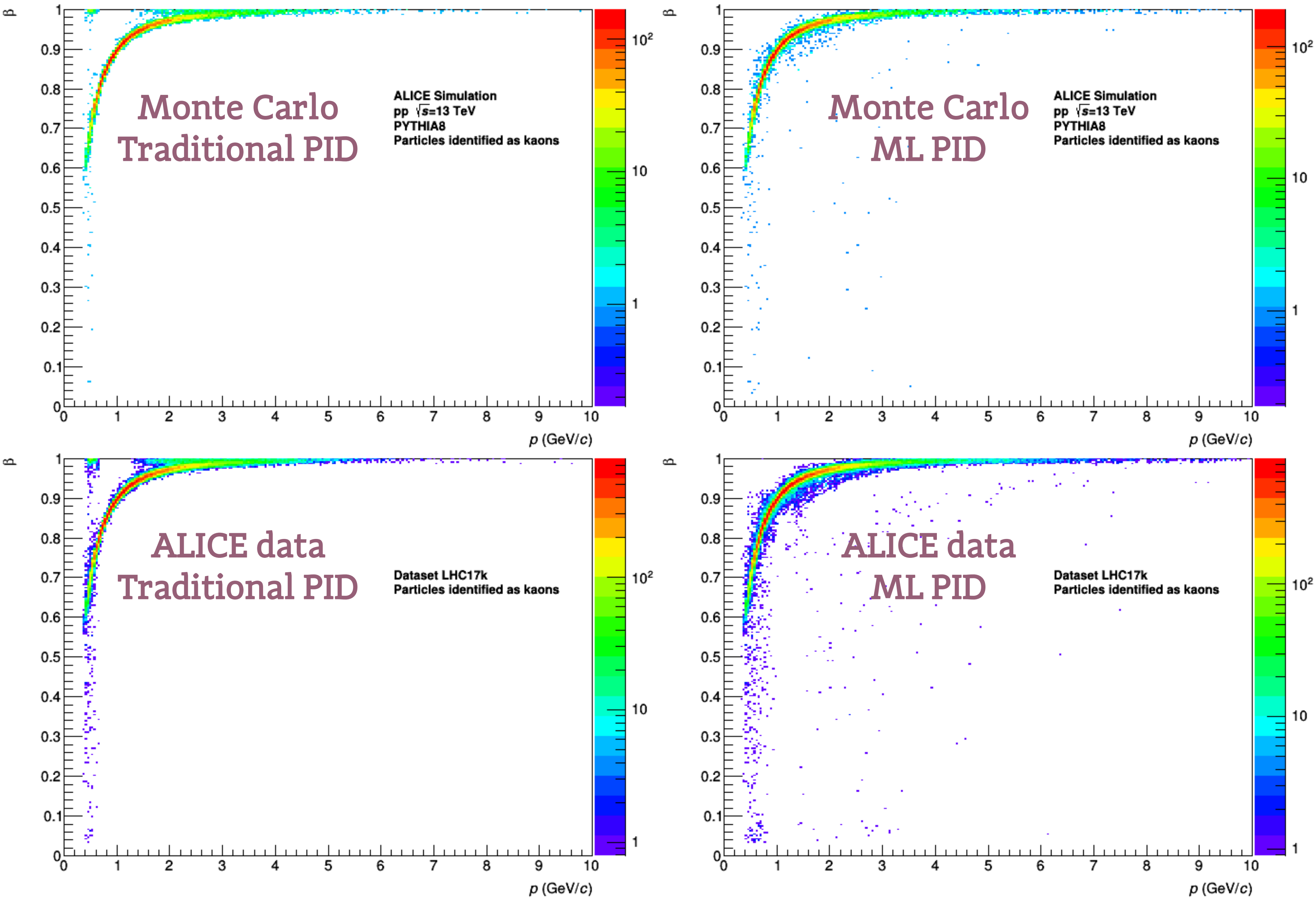


Summary

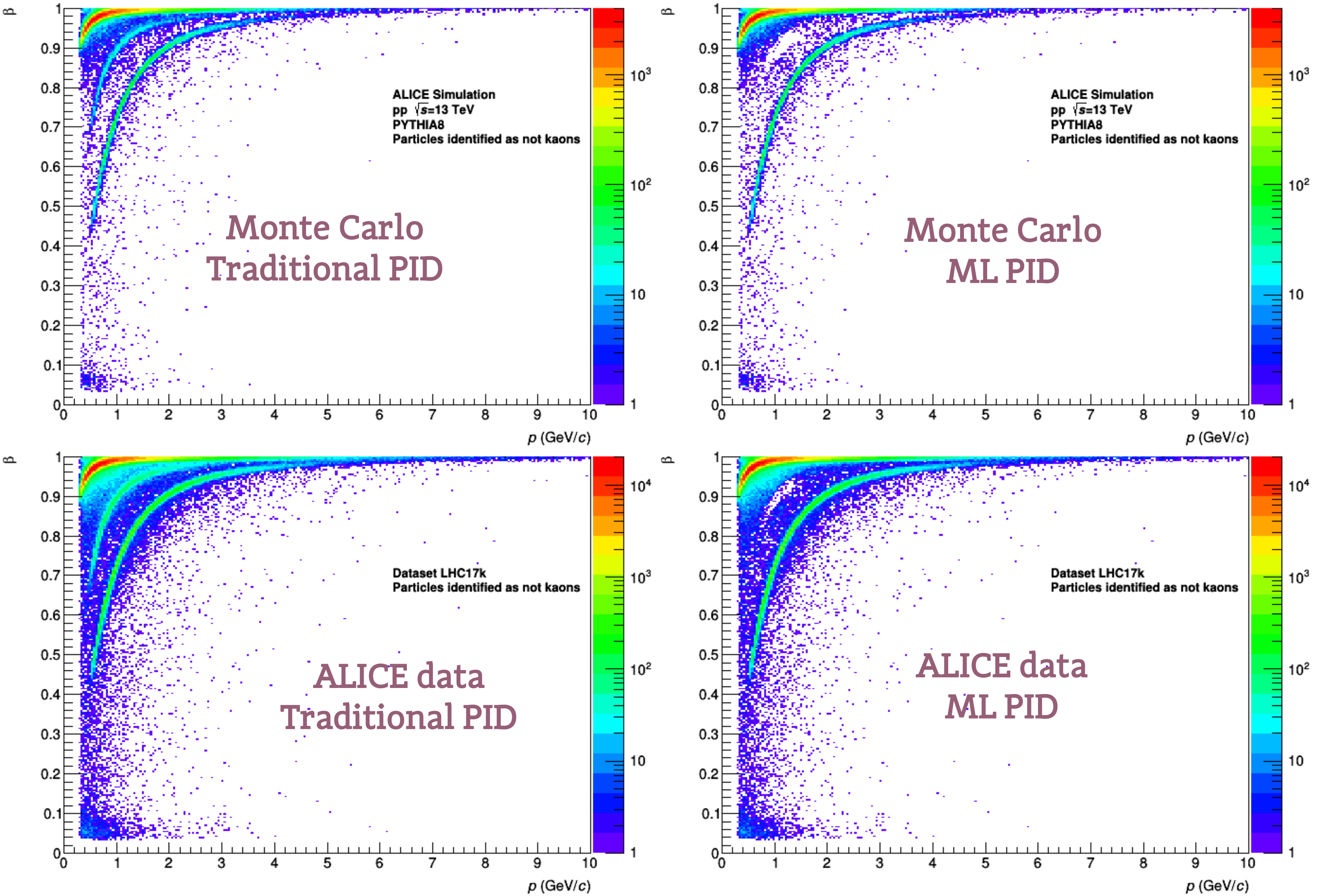
- ML-based PID outperforms traditional PID, especially in the low momentum region
- Training needed only once for each data set – no need for manual cut optimizations
- Quality of final classification more vulnerable to discrepancies between MC and real data
- Domain Adaptation techniques look very promising → hope to deliver working interface in O^2
- **Problems encountered in preliminary work:**
  - track-by-track implementation in AliRoot (optimal from our side) is very slow
  - C++ <-> Python connection is also a weak point

Backup

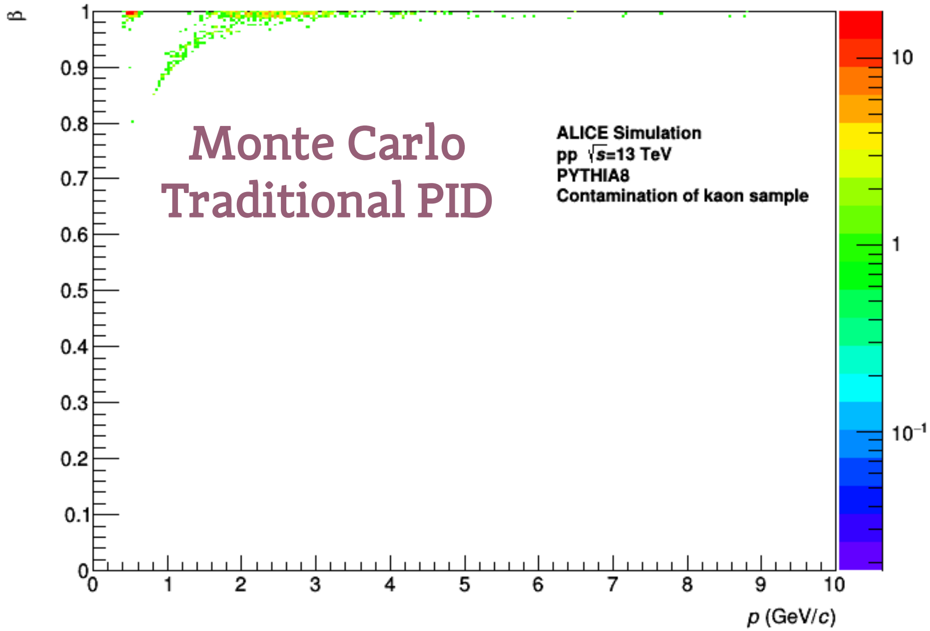
TOF accepted kaons

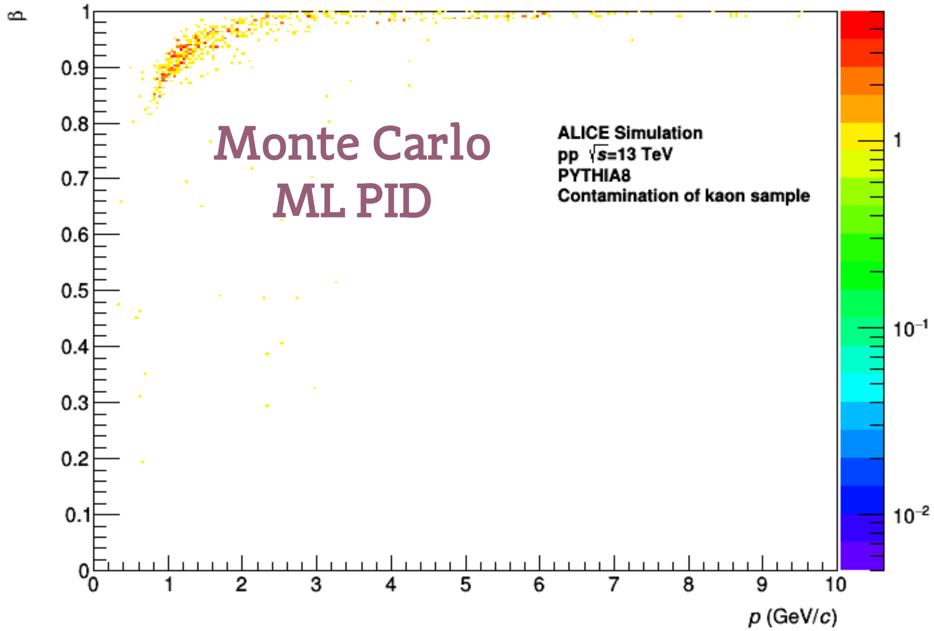


TOF rejected (not kaons)



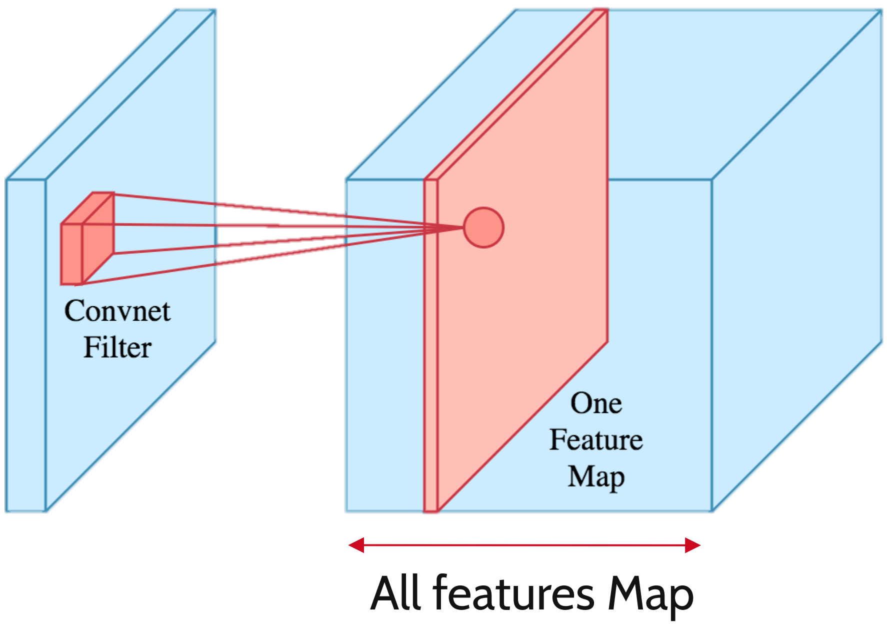
TOF contamination in kaons





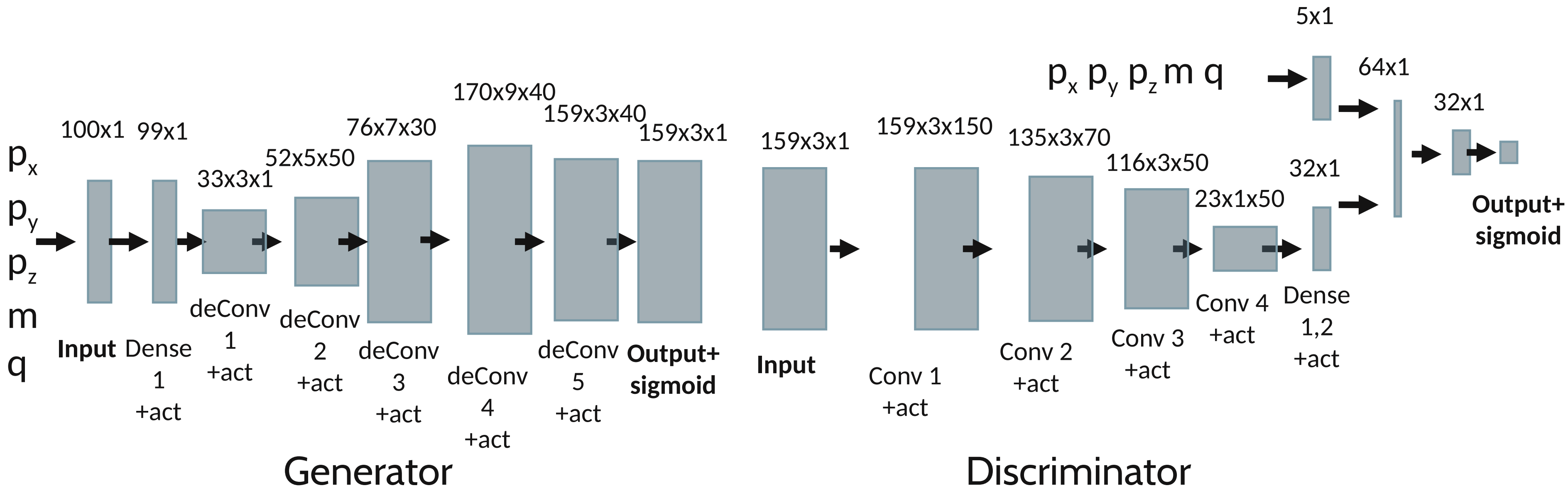
Deep Convolutional GAN

- Class of architectures which use the convolutional tools and deconvolutional layers – mostly used with images



condDCGAN: Conditional DCGAN

- Generator – deconvolutional layers
- Discriminator – convolutional layers
- Network conditioned on particle momenta, mass, and charge
- Output classification – sigmoid function



condDCGAN+: combined loss

- Training on on the full MC simulations
- Preparing the noise from initial parameters of MC simulations
- Comparing the generated samples with original ones
- Combining origininal conditional GAN loss with the results of comparison

$$\mathcal{L}_G(m, X) = \mathbb{E}_{z \sim p_z(z|m)}[\alpha \log(1 - D(G(z))) + \beta \frac{1}{n} \sum_{i=1}^{n} (X_i - G(\hat{z})_i)^2]$$

m - initial parameters (particle momenta),
X - original value corresponding to m ,
p(z|m) - distribution of a noise vector under initial parameters m
z - input into a generator
G and D - generator and discriminator
n - the number of produced clusters

Additional parameters α and β are used to weight the share of individual losses.
Best performing values are α = 0.6 and β = 0.8

Other areas of research

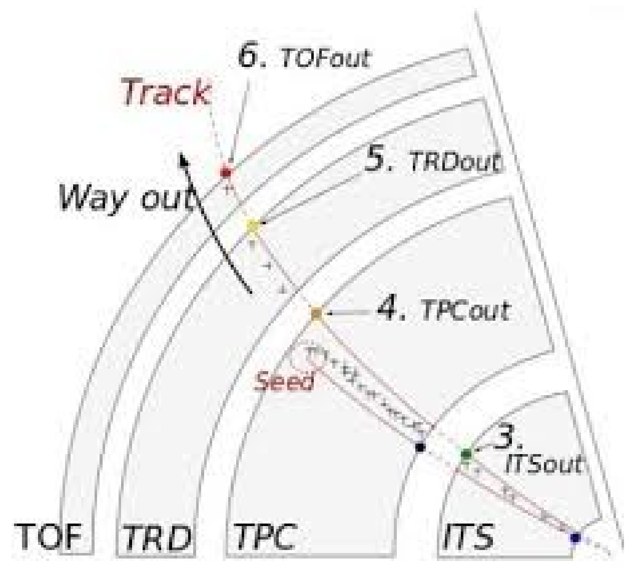
- Data Quality Assurance – prediction of detector quality label assignment

- Simulation of TPC clusters in Monte Carlo data using generative networks
  → next slides

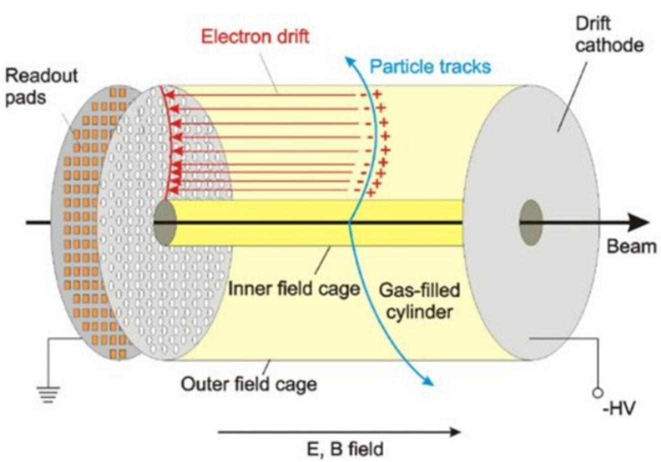
Simulation of TPC clusters in Monte Carlo data using generative networks

Time Projection Chamber

- Tracking in ALICE is performed by ITS, TPC, TRD and TOF
- First attempts – focus on the TPC only:
  - main tracking device
  - located from 0.8 m (inner radius) to 2.5 m (outer radius) from the beam and extending ~2.5 m in each direction along the beam axis
  - volume of 95 m^3
  - filled with Ne-CO_2 gas mixture (90%-10%)
  - **clusters** - points in 3D space, together with the energy loss, which were presumably generated by a particle traveling through
  - provides up to 159 clusters per track



ALICE Data Preparation Group



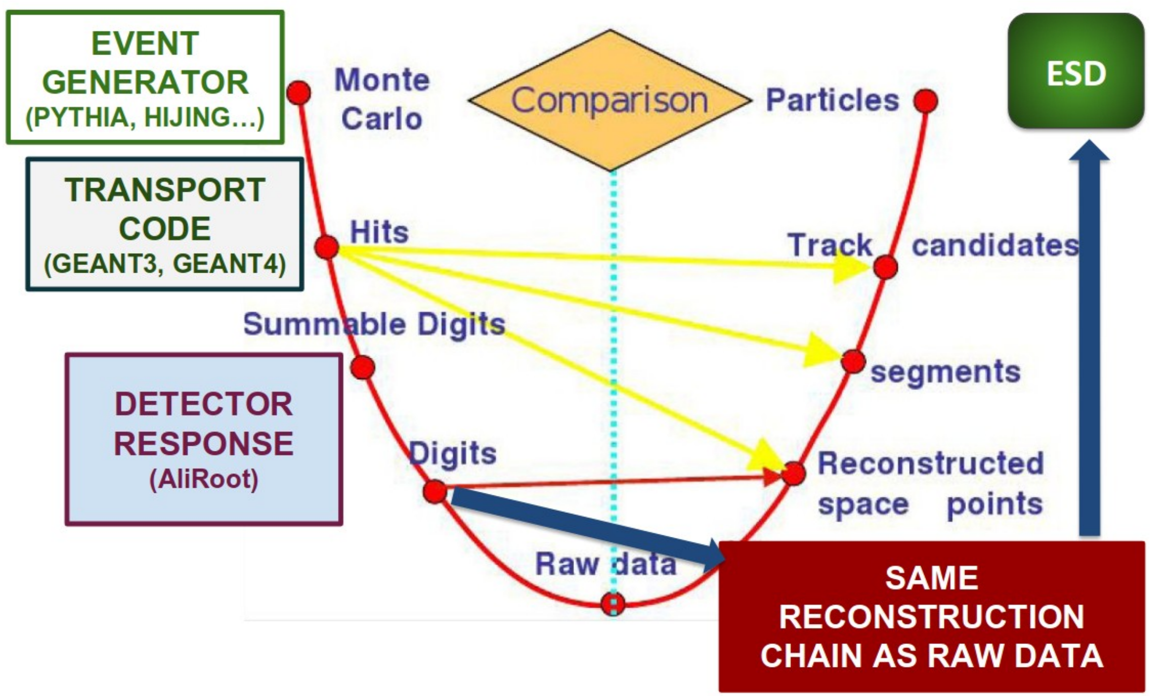
I.Konorov, Front-end electronics for Time Projection chamber

Simulation and reconstruction

- Current process relies on 5 independent modules
- The computationally most expensive module is particle propagation through the detector's matter

Collision generator → Particle propagation → Electronic signals (digits) → Digits to Clusters → Tracking

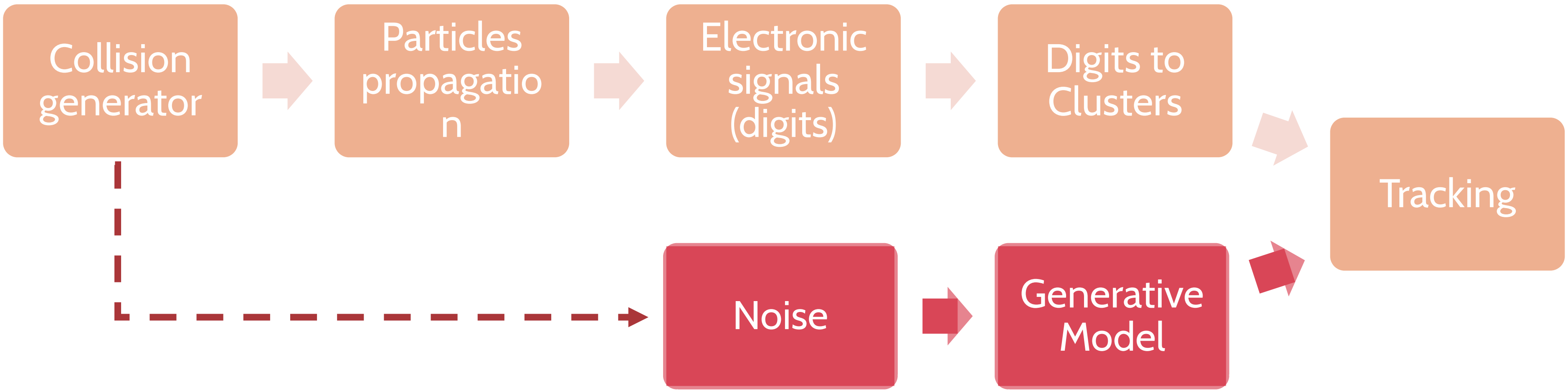
Monte Carlo simulation



Real data

ALICE Data Preparation Group

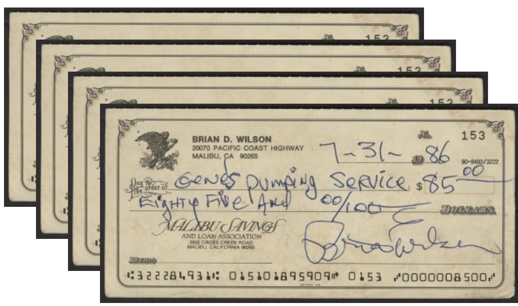
Simulation and reconstruction



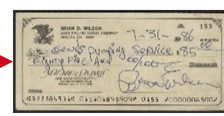
- **Generative solution for cluster simulation:**
  - substitute the detector simulation and check for the speed-up
  - full simulation **still needed** to generate training samples
  - **immediate drawback:** quality of such MC data can be either comparable or lower than the full detector simulation – limits potential applications

Generative Adversarial Networks

- Generative Adversarial Network (GAN) is a neural network architecture of two networks competing with each other (playing a min-max game)
  - **"Generator"** is trained to produce fake data resembling the real data
  - **"Discriminator"** aims to predict whether an example data is real or fake



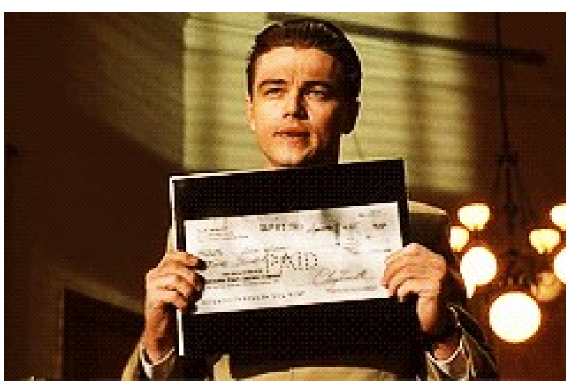
https://33milesinnewaygocounty.files.wordpress.com

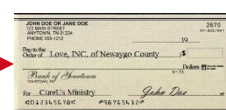


Discriminator

https://thechive.files.wordpress.com

Generator



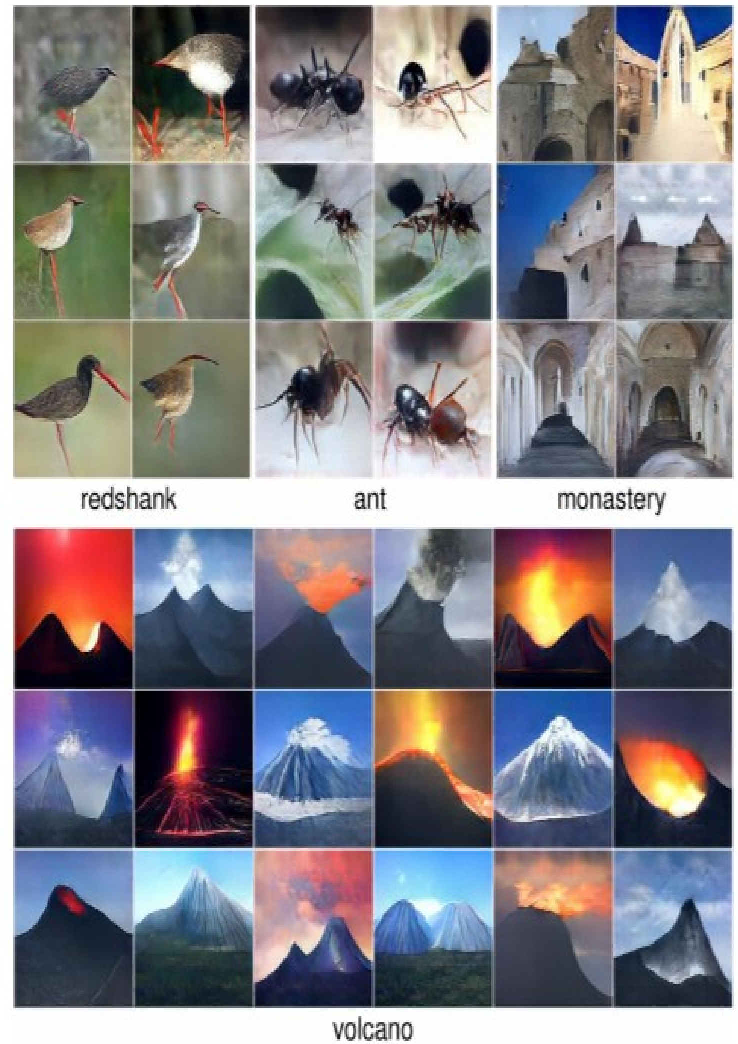


https://giphy.com/gifs/leonardo-dicaprio-catch-me-if-you-can-5leocharacters-t1h4nnWEWKfn2

Generative Adversarial Networks

- Typical use cases:
  - mainly generation of photo quality fake images (i.e. of celebrities)

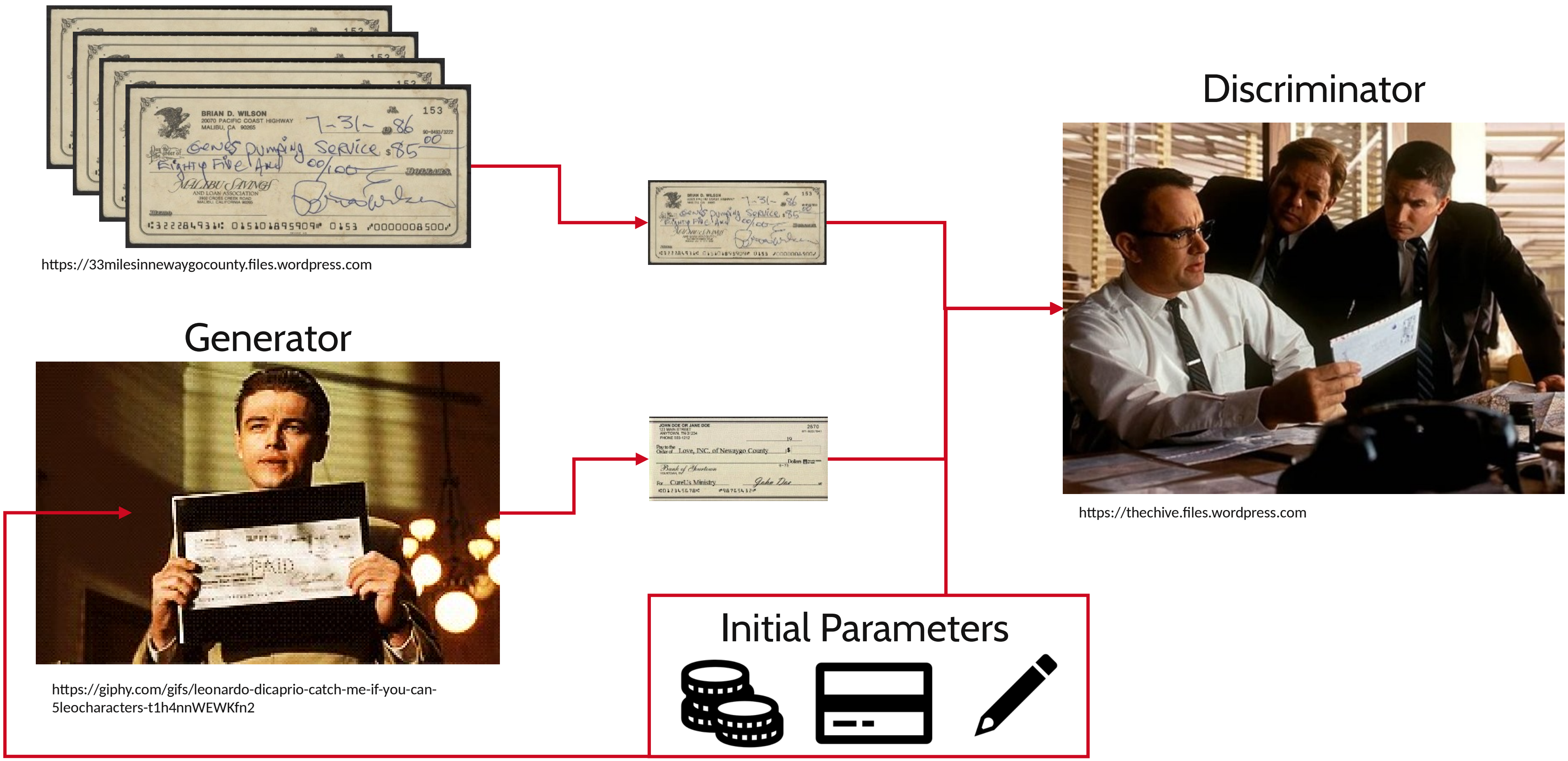
https://arxiv.org/abs/1710.10196



https://arxiv.org/pdf/1612.00005v1.pdf

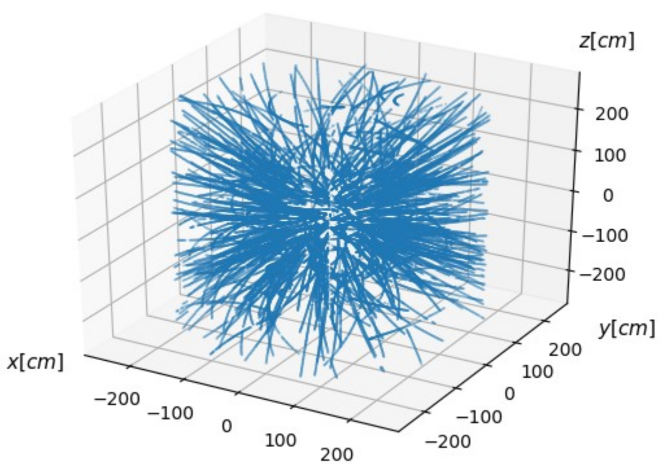
Generative Adversarial Networks

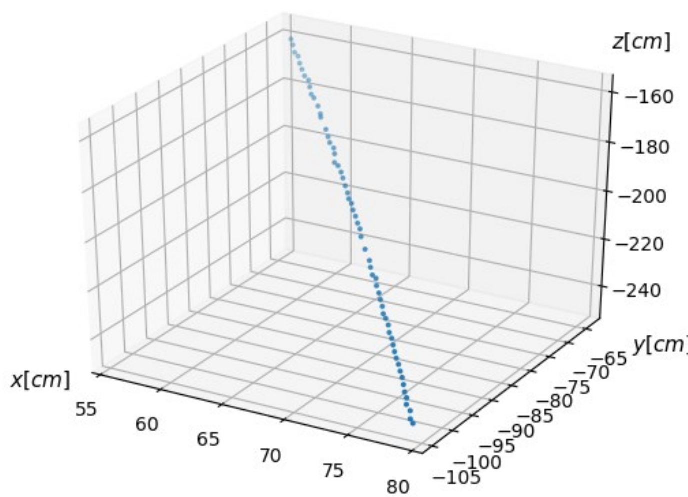
- **Extending the GAN architecture – provide a set of initial parameters for the generator and discriminator:**
  - generator would not generate a random output, but a customized one
  - in our case: initial momenta of Monte Carlo particles



TPC clusters with GANs

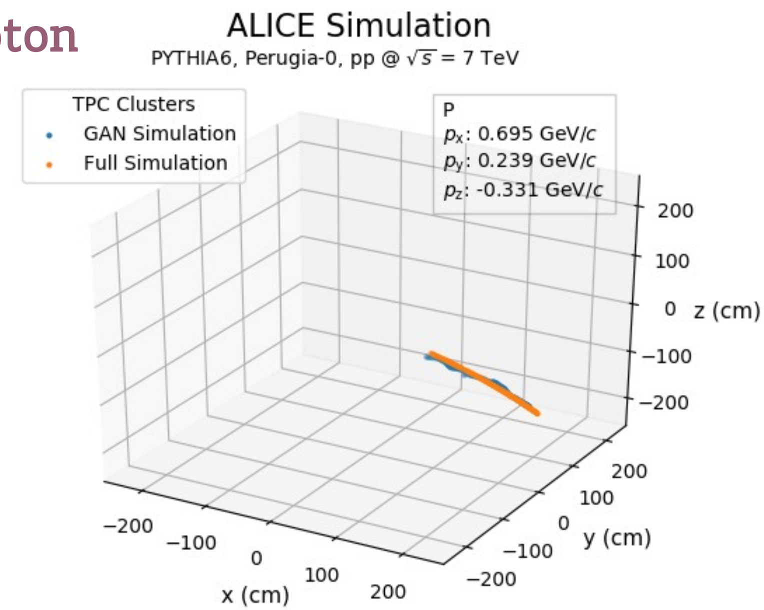
- It is not (yet!) possible to generate the full 3D image of the event at once (especially in the Pb-Pb event)
- Our solution is to:
  - generate clusters for single particles
  - two separate flows for spatial coordinates (x,y,z) and the energy
  - in the beginning focus only on 3D coordinates
  - merge generated samples (individual particles) into full images
  - training of the GAN on original full simulations



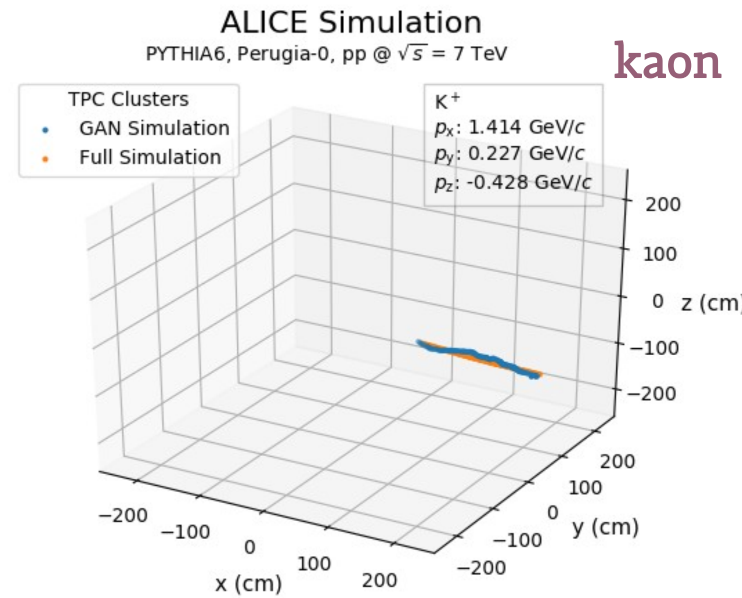


Example results

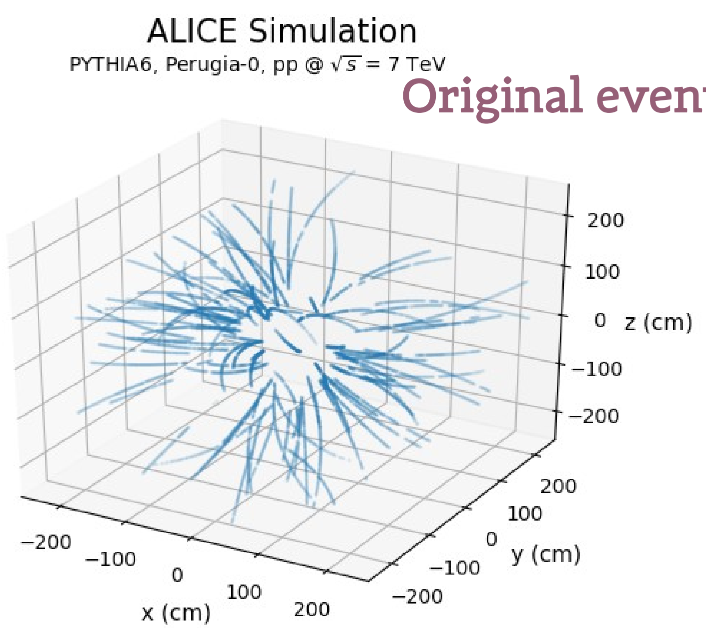
proton



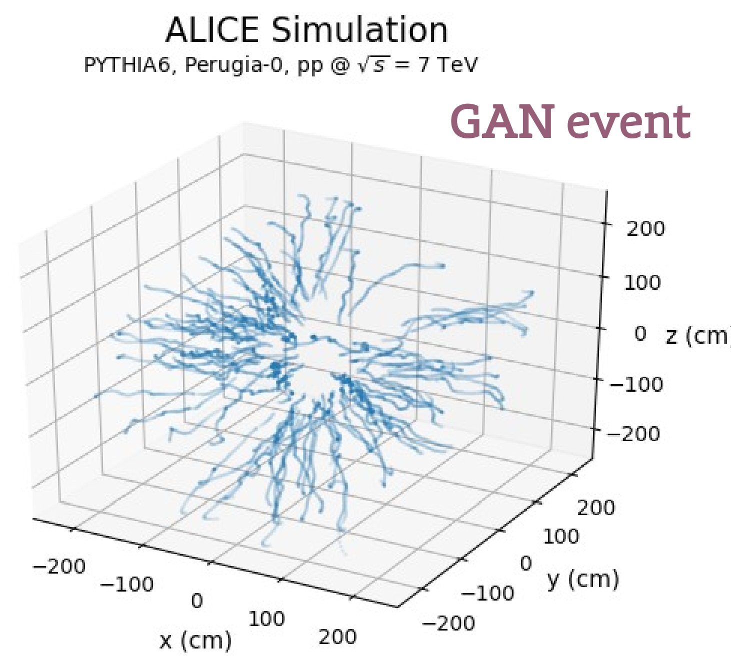
kaon



Original event



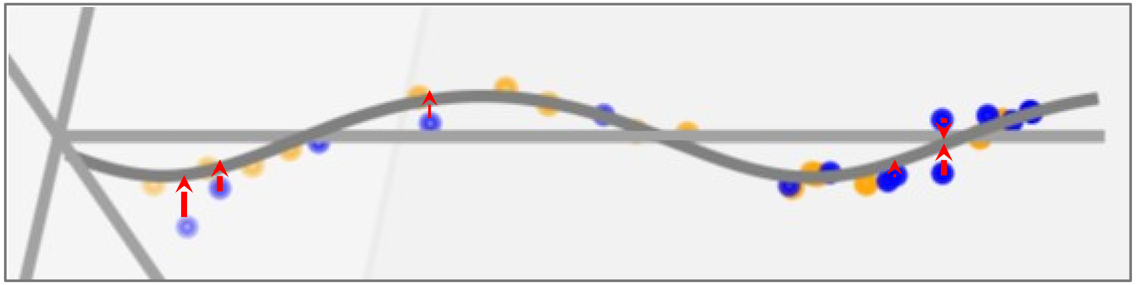
GAN event



Results

- Mean Squared Error (MSE) from the original helix as a quality measure
- Evaluation conducted on the separate test-set with ~15000 tracks

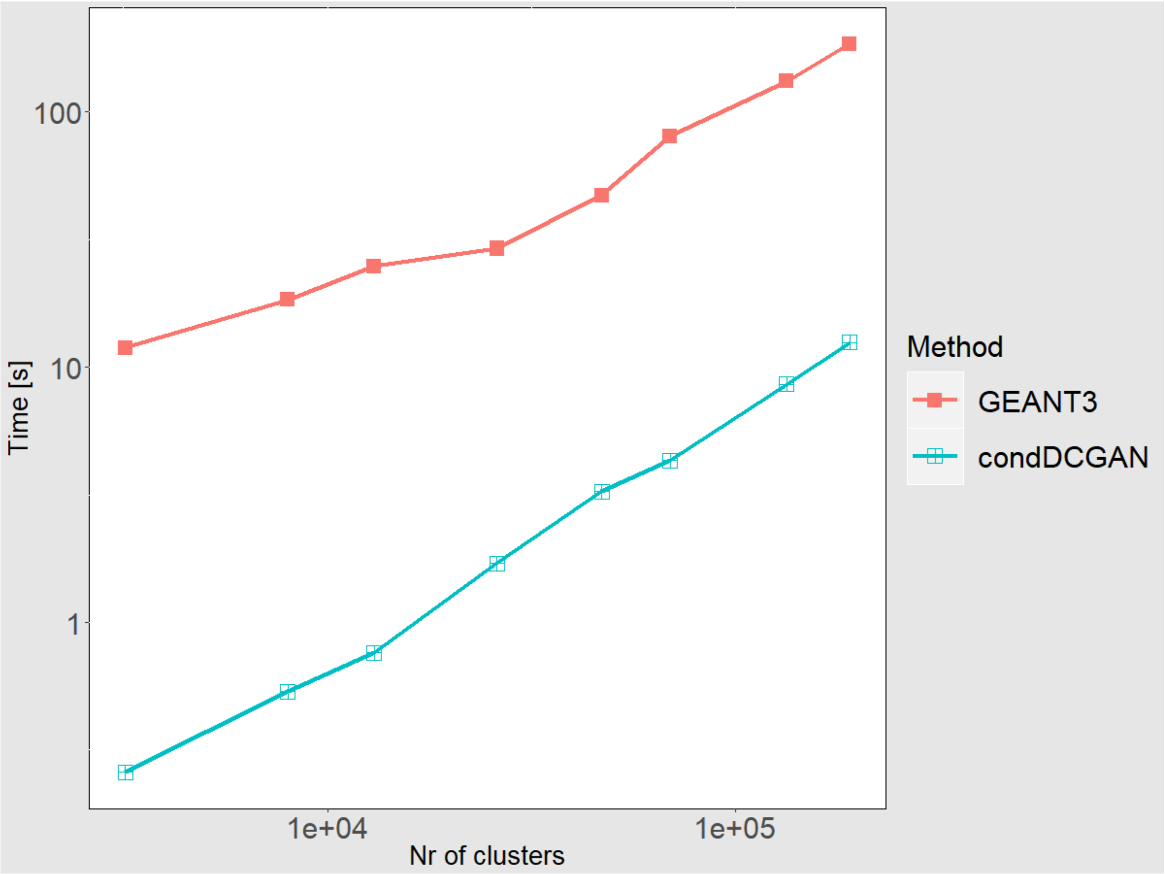
MSE visualisation:
Red - error
Grey- ideal helix
Orange – original clusters
Blue – generated clusters



Method	Mean MSE (mm)	Median MSE (mm)	Speed-up
GEANT3	1.20	1.12	1
Random (estimated)	2500	2500	N/A
condLSTM GAN	2093.69	2070.32	100
condLSTM GAN+	221.78	190.17	
condDCGAN	795.08	738.71	25
condDCGAN+	**136.84**	**82.72**	

Computational cost

- Performance test conducted on the standalone machine with Intel Core i7-6850K (3.60 GHz) CPU using single core and no GPU
- Additional order of magnitude speed-up for GAN models with nVidia Titan Xp GPU



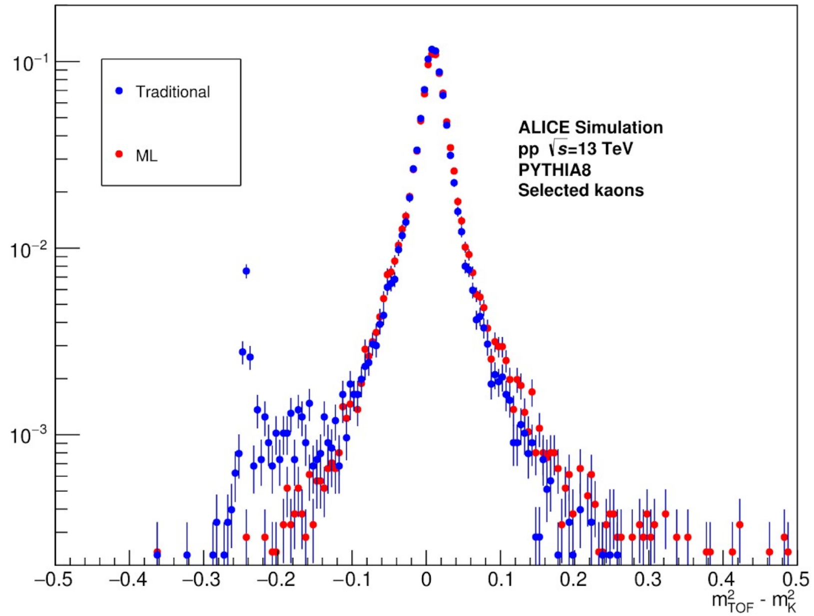
TOF time

- From our point of view TOF has a fantastic feature of a possibility to calculate mass of the recorded particle and compare it to the one from PDG

$$m_{TOF}^2 = p^2\left(\frac{1}{\beta} - 1\right)$$

- Thanks to that we can test contamination independently of MC simulations

Monte Carlo



ALICE data